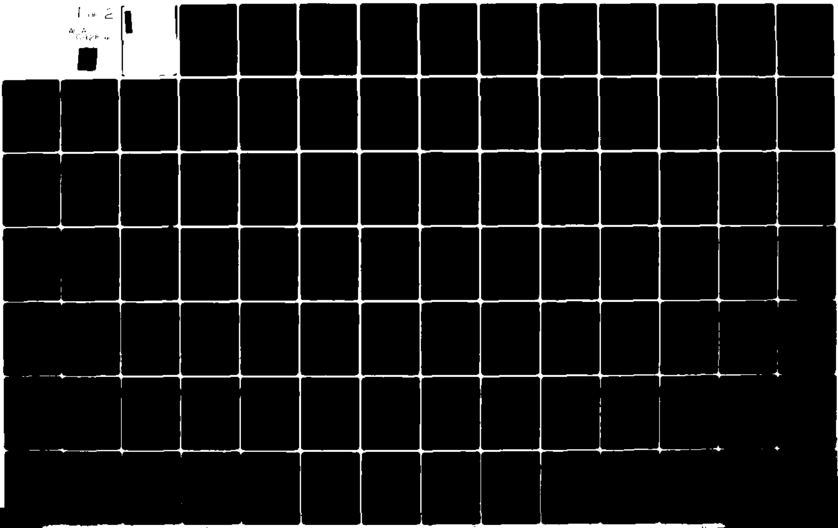
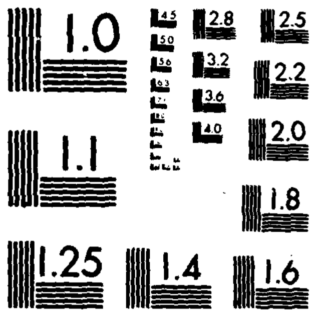
AD-A092 696 NAVAL RESEARCH LAB WASHINGTON DC F/G 9/2
ABSTRACT INTERFACE SPECIFICATIONS FOR THE A-7E DEVICE INTERFACE--ETC(U)
NOV 80 R A PARKER, K L HENINGER, D L PARNAS
UNCLASSIFIED NRL-MR-4385 NL

1 of 2
AD A
092696




1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(9) **REPORT DOCUMENTATION PAGE**	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: NRL Memorandum Report 4385	2. GOVT ACCESSION NO.: AD-A092 696
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ABSTRACT INTERFACE SPECIFICATIONS FOR THE A-7E DEVICE INTERFACE MODULE	5. TYPE OF REPORT & PERIOD COVERED: Interim report on a continuing NRL problem.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Robert A. Parker, Kathryn L. Heninger, David L. Parnas* and John E. Shore	8. CONTRACT OR GRANT NUMBER(s): [illegible]
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Research Laboratory, Washington, DC 20375	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ~~63721~~ ZF21242001, 75-0106-0-1
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Research Laboratory, Washington, DC 20375	12. REPORT DATE: November 20, 1980
	13. NUMBER OF PAGES: 176
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

*On leave from the University of North Carolina, Chapel Hill

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Specifications
Real-time systems
Software documentation
Software engineering
Abstract interfaces

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
As part of the experimental redesign of the flight software for the Navy's A-7 aircraft, software modules were designed to encapsulate the characteristics of hardware devices connected to the computer. The purpose of these device interface modules is to allow the remainder of the software to remain unchanged when devices are changed or replaced. To achieve this purpose, the modules were designed according to the abstract interface principle, documented according to a standard organization and reviewed by a systematic procedure based on the properties expected of abstract interfaces.

(Continues)

20. Abstract (continued)

This report contains (a) an explanation of the abstract interface approach, (b) a description of the standard organization for interface specifications, (c) a description of the review procedure, and (d) interface specifications for all the device interface modules in the A-7 software.

As well as serving as development and maintenance documentation for the A-7 redesign, this document is intended to serve as a model for other people interested in applying the abstract interface approach on other software projects.

Preface

The Operational Flight Program (OFP) for the Navy's A-7 aircraft is considered a successful program: it works reliably. However it is expensive to maintain because it has problems typical of much DoD software:

* it barely meets its time and space limitations,

* it is not fully understood by the maintenance personnel,

* it is poorly documented, and

* it is difficult to change.

Although various software engineering techniques have been proposed to deal with such problems, there is a widespread reluctance to modify or abandon current techniques which, despite the problems mentioned above, are the basis for many acceptable programs such as the A-7 OFP. Two reasons for this reluctance are:

* No-one has proven that the techniques are useful in the DoD context of complicated requirements and stringent resource limitations.

* There are no fully worked out examples that system developers can use as models in applying the new techniques to DoD sytems.

In order to demonstrate feasibility and to provide a useful model, the Naval Research Laboratory (NRL) and the Naval Weapons Center (NWC) have embarked on a joint project in which certain software engineering techniques will be used to redesign and rebuild the A-7 OFP. Among the techniques to be used are the following:

* modularity and information hiding (/1/,/2/)

* formal specifications (/3/-/6/)

* abstract interfaces (/7/)

* cooperating sequential processes (/8/,/9/,/14/)

* process synchronization routines (/8/,/10/)

* resource monitors (/11/-/14/)

The first product of the project was /16/. It describes the externally visible behavior required of the A-7 OFP without describing an OFP implementation.

The redesigned software is divided into modules according to the information-hiding principle /2/. For each module, we will provide two types of documentation: interface specifications, showing the externally visible characteristics of the module, and abstract programs, showing the internal implementation decisions made for the module.

This report is the second published product of the project. It includes interface specifications for the device interface modules, which encapsulate the characteristics of hardware devices connected to the computer.

This report is intended to serve as an example of good module documentation. It describes module interfaces without giving away implementation details. It demonstrates the design of abstract interfaces /7/ for hardware devices, showing some of the design problems we faced and how we solved them. It also demonstrates the design of a systematic procedure for reviewing module interface designs.

Preface References

/1/ Parnas, D. L.; "Information Distribution Aspects of Design Methodology"; Proceedings of IFIP Congress 71; North Holland Publishing Company, Amsterdam, New York, Oxford, TA-3, pp. 26-30.

/2/ Parnas, D. L.; "On the Criteria To Be Used in Decomposing Systems into Modules"; Comm. ACM, Vol. 15, No. 12 (December 1972), pp. 1053-1058.

/3/ Parnas, D. L.; "The Use of Precise Specifications in the Development of Software"; Proceedings of the IFIP 1977; North Holland Publishing Company, Amsterdam, New York, Oxford, pp. 861-867.

/4/ Liskov, B. and Zilles, S.; "Specification Techniques for Data Abstractions," IEEE Transactions on Software Engineering, Vol. SE-1, No. 1 (March 1975).

/5/ Liskov, B. and Berzins; "An Appraisal of Program Specifications" in Wegner, P. Research Directions in Software Technology, to be published by MIT Press, Fall 1978.

/6/ Parnas, D. L.; and Handzel, G.; More on Specification Techniques for Software Modules; Fachbereich Informatik, Technische Hochschule Darmstadt, (February 1975).

/7/ Parnas, D. L.; Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems; NRL Report No. 8047 (1977).

/8/ Dijkstra, E. W.; "Co-operating Sequential Processes"; F. Genuys (ed.), Programming Languages, Academic Press, New York (1968), pp. 43-112.

/9/ Shaw, Alan C.; The Logical Design of Operating Systems; Prentice-Hall, Inc., Englewwood Cliffs, NJ (1974).

/10/ Lipton, R; On Synchronization Primitive Systems; Ph.D Disertation, Carnegie-Mellon University (1973). Updated version available as a Computer Science Research Report from Yale.

/11/ Hoare, C.A.R.; "Monitors: an Operating System Structuring Concept"; Comm. ACM, Vol. 17, No. 10 (October 1974), pp. 549-557.

/12/ Brinch Hansen, P.; Operating Systems Principles; Prentice-Hall, Englewood Cliffs, NJ (1973).

/13/ Howard, J.; "Proving Monitors"; Comm. ACM, Vol. 19, No. 5 (May 1976), pp. 273-279.

/14/ Parnas, D. L., et al; "Implementing Processes in HAS"; in Software Engineering for Software Acquisition Managers; Naval Research Laboratory, Washington, D.C. (July 1977) Document HAS.9.

/15/ Parnas, D. L.; "Designing Software for Ease of Extension and Contraction"; Proceedings of the 3rd International Conference on Software Engineering (10-12 May 1978), pp. 264-277.

/16/ Heninger, K., et al, Software Requirements for the A-7E Aircraft, NRL Memorandum Report 3876; Nov. 1978.

Acknowledgements

We would like to thank all the people at the Naval Weapons Center who helped us, but especially the following people who answered questions patiently and reviewed the document carefully.

Fritz Artz
Jack Basden
Richard Breisch
Woody Chartier
Sandra Fryer
Charlie Hill
Mark Jacobson
Dell Kellogg
Joe Law
Forest Lloyd
Mike Maxfield
Lee Thomson
Don Walters (Vought Corp.)
Bob Westbrook
Jack Willbur
Jack Williams

We would also like to thank David Weiss, Paul Clements, John Fredrickson, and Preston Mullen for their comments and assistance during the preparation and review of this document.

TABLE OF CONTENTS

DEVICE INTERFACE SPECIFICATIONS

DI.7 FORWARD LOOKING RADAR

The Forward Looking Radar (FLR) is used to measure distances and to display a point on a radar screen. It operates very differently in its four mutually exclusive modes, which are called RANGING, CURSOR, TF, and IDLE. In RANGING mode, the FLR is a sensor that measures the slant range from the aircraft to a given ground location. In CURSOR mode, the software controls the position of two cursors on the radar display screen. In TF mode, no software actions have any effect on the FLR. In IDLE mode, the FLR can be directed to perform a built-in test and to report the results.

DI.8 HEAD-UP DISPLAY

The Head-Up Display (HUD) is a display device that projects information into the pilot's field of vision so that he can see it without looking around the cockpit. The HUD displays two kinds of symbols: location indicators, which can be positioned under software control, and value indicators, which are always in the same place and display a value provided by software.

DI.9 INERTIAL MEASUREMENT SET

The Inertial Measurement Set is a sensor that can be used to measure aircraft attitude and velocity.

DI.10 PANEL

The Panel is a data entry and display device. The data entry component consists of a keyboard that can be used to enter digits and letters. The display component has a few special purpose indicators and three windows that can be used to display alphanumeric strings.

DI.11 PROJECTED MAP DISPLAY SET

The Projected Map Display Set is a display device that shows a map of the area surrounding a specified geographic location. The specified location can be positioned under one of two reference points on the map display screen. Maps of more than one scale may be displayed. Scale and position selection are under software control.

DI.12 RADAR ALTIMETER

The Radar Altimeter is a sensor that measures the altitude of the aircraft above the local terrain, either land or water.

DI.13 SHIPBOARD INERTIAL NAVIGATION SYSTEM

The Shipboard Inertial Navigation System is a sensor that reports the position, velocity, and attitude of a ship located near the aircraft.

APPENDICES

ABSTRACT INTERFACE DESIGN

MOTIVATION

Embedded real-time software systems usually have complex and restrictive external interfaces. Since embedded software is usually a small component of a much larger system, the interfaces are seldom modified for the convenience of the software designer. The A-7 avionics software is typical: twenty-one devices are connected to the computer, including sensors, displays, and equipment controlled by the computer. Arbitrary interface characteristics, such as value encodings and timing quirks, are subject to change both during development and after initial deployment. Inadequacies may be discovered in the device specifications; a supplier may deliver a device that is judged adequate even though it does not exactly meet its specifications; a device may be replaced by an improved device; or new connections may be added between devices.

It is a common but undesirable property of embedded software that a change in a device interface requires widespread changes to the software because many programs are based on arbitrary interface details. If an interface changes, programs depending on it become invalid. Because these dependencies are seldom explicitly documented, interface changes often have surprising ramifications.

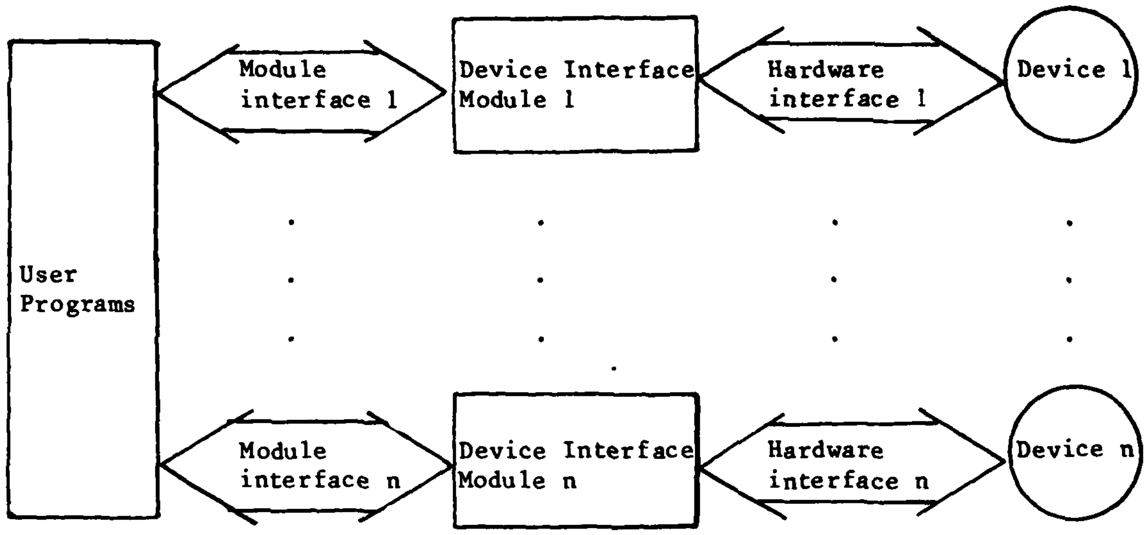
To avoid these problems we divide the software into two groups of components: 1) the _device interface modules_ containing the device-dependent code, and 2) the device-independent remainder of the software, including the _user programs_, so called because they use the device interface modules. Device interface modules provide _virtual devices_, that is, device-like capabilities that are partially implemented in software. This software structure is illustrated in figure 1.

Design of device interface modules has the following goals:

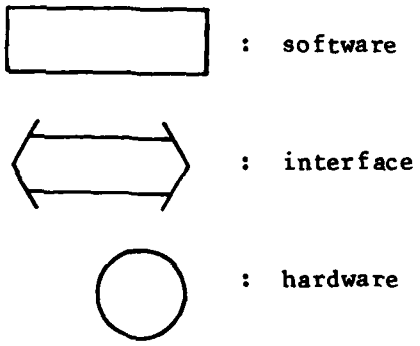
• _Confining changes_: Designing device interface modules is a special case of the information-hiding approach (ref (6)); hardware interface details are hidden within modules that should be the only system components requiring changes when devices are modified or replaced by others that can perform the same basic functions. Problems in confining change are caused by three types of errors:

1) The device interface module allows user programs to exploit special characteristics of a particular device so that user programs must be revised if the device is replaced.

Figure 1: Software Designed with Device Interface Modules



<u>Legend</u>:



2) The virtual device lacks essential capabilities, so that user programs must access the actual device directly; again user programs must be revised if the device is replaced.

3) Programs that are not necessarily device-dependent are included in the device interface module. As a result, the device interface module may need to be changed if the requirements change even if the device is not changed. Furthermore the module will be harder to change when the device is changed.

In summary, a device interface module will ideally

1) be the only component that needs to change if a device is changed;

2) not need to change unless the device is changed; and

3) be relatively small and straightforward so that it is easy to change.

• Simplifying the rest of the software: Embedded software is often hard to understand because its correctness depends on many arbitrary interface details. If these details are confined to device interface modules, then user programs should be simpler, easier to write correctly, and easier to understand than if they used the hardware interfaces directly.

• Enforcing disciplined use of resources: Software reliability is enhanced when all programs that access a device adhere to certain disciplines, such as regular checks for undesired events (ref (7)) and standard protocols for device sharing (ref (8)). If these disciplines are built into the device interface modules, they are systematically enforced; programmers writing user programs need not be concerned with them.

• Code sharing: When many programs access a device directly, they often contain similar subprograms performing the same device control functions. With device interface modules, this code need only be written once, saving programming, debugging, and testing time, and possibly computer storage.

• Efficient use of devices: Independently written programs often cause devices to repeat actions unnecessarily. Centralizing device-access code should make it easier to avoid unnecessary operations.

To achieve these goals and to avoid the mistakes mentioned earlier, the interface between a device interface module and user programs must be an abstract interface, as defined in the next section.

DEFINITIONS

· Interface: The interface between two programs consists of the set of assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his own program. For convenience, we use the phrase "assumptions made by program A about program B," to mean the properties of B that must be true in order for A to work properly. These assumptions are not limited to the calling sequences and parameter formats traditionally found in interface documents; they include additional information such as the meaning and limits of information exchanged, restrictions on the order of events, and expected behavior when undesired events (ref (7)) occur. There is an analogous definition of the interface between a program and a device.

· Abstraction: An abstraction of a set of objects is a description that applies equally well to any one of them. Each object is an instance of the abstraction. For a non-trivial abstraction, there is a one-to-many relationship between the abstraction and the objects it describes. Differential equations are an example of a mathematical abstraction representing systems as diverse as electrical circuits and collections of springs and weights.

An abstraction that is appropriate for a given purpose is easier to study than the actual system because it omits details that are not relevant for that purpose. A road map is an abstraction used to represent a road network; the graph represents the directions, relative lengths, and intersections of roads, but it does not show whether a road is made of asphalt or how it is banked. It is far easier to find a good route by studying a road map than by exploring the actual roads.

Any result obtained by studying an abstraction can be applied to any system represented by the same abstraction. Well-known graph theoretic results can be applied to a road map to determine the shortest route; the same methods have been applied to solve a wide variety of problems in other systems represented by directed graphs. Results may be misleading if they are obtained from an inappropriate abstraction, i.e., one that omits relevant details. For example, a road map is not sufficient to find the quickest route because it does not show other factors affecting driving time such as speed limits.

· Abstract interface: An abstract interface is an abstraction that represents more than one interface; it consists of the assumptions that are included in all of the interfaces that it represents. An abstract interface for a given type of device reveals some, but not all, properties of the actual device: it describes the common aspects of devices of that type, omitting the aspects that distinguish them from each other.

• Device interface module: A device interface module is a set of programs that translate between the abstract interface and the actual hardware interface. The implementation of this module is possible only if all assumptions in the abstract interface are true of the actual device.

• Secret: Secrets of a device interface module are assumptions about the actual device that user programs are not allowed to make. The secrets are information about the current device that need not be true of other devices with the same functions. Secrets must be taken into account somewhere in correctly working software; they are encapsulated in a device interface module.

• Undesired event assumptions: The interface between programs A and B includes both the assumptions made by A about B and the assumptions made by B about A. Systems can be designed so that only one of two programs relies on the other meeting its specifications. A program can be designed so that it does not rely on user programs using it correctly; it can check for improper uses and signal undesired events when they occur. However, the error checking and reporting require extra instructions. In development versions of the A-7 software, the device interface modules will assume that undesired events can occur; they will contain code to check for errors made by user programs. In the production version there will not be room for that error-checking. The device interface modules will assume that improper uses will not occur; the error-checking code will be omitted to make the system smaller and faster. If a problem occurs during operation of the production system, the error-checking will be reinserted to help locate the cause. The software will be written in such a way that the error-checking code can be easily included or omitted when the program is assembled. This applies only to programming errors; error checks specified in the software requirements (ref (1)) will never be omitted.

• Access functions: An access function is a program that is part of one module and may be called by programs in other modules. There are different kinds of access functions; some return information to the caller; others change the state of the module to which they belong.

• Events: Events are signals from a module to user programs indicating the occurrence of some state change within the module. They resemble hardware interrupts because they occur at unpredictable times and are not synchronized with the control flow of the user programs. In the A-7 system, modules will use the eventcount mechanism (ref (4)) to signal the occurrence of an event to user programs that are waiting for it to occur.

DESIGN APPROACH

This section describes an approach to the design of abstract interfaces. The approach is based on obtaining two partially redundant descriptions of the interface.

· DESCRIPTION 1: ASSUMPTION LIST CHARACTERIZING THE VIRTUAL DEVICE.

For an application area such as avionics, many devices fall into standard types; all devices of a given type have many common characteristics. For example, as shown by advertisements in Aviation Week and Space Technology, computer panels vary little in the features seen by the pilot. For each hardware device, make a list of the characteristics that are not likely to change if the device is replaced by another device of the same type. To do so requires considerable study of devices that are available or being developed. The list of common characteristics is a description of the assumptions that user programs are allowed to make about the virtual device. The assumptions characterize device capabilities, modes, information requirements, behavior, and proper use of the device. A typical assumption might be:

"The device provides information from which barometric altitude can be determined."

We are quite certain that only devices satisfying this assumption will replace the current barometric altitude sensor. Note that this assumption does not describe the form of the information, which may vary from one device to another.

Many assumptions will appear innocuous, but they must be recorded anyway. During the A-7 design reviews, some seemingly innocuous assumptions were found to be false.

· DESCRIPTION 2: PROGRAMMING CONSTRUCTS EMBODYING THE ASSUMPTIONS.

The second description specifies the access functions and events that can be used by user programs. The access functions can be called by user programs to access the data or facility provided by the virtual device. For example, an interface might provide an access function "GET_BAROALT", which returns a barometric altitude value. For each access function, we specify the values returned, the limitations, and the effect it has on the virtual device to which it belongs. User programs can also use the events in order to be signalled when the virtual device changes state. For example, user programs may need to be signalled when a virtual sensor is no longer operational.

Why two descriptions?

These two descriptions are partially redundant, i.e., the specifications for the programming constructs imply the assumptions. For example, specifications for the access function "GET_BAROALT" imply the assumption that the device provides information from which barometric altitude can be determined. The access function specifications provide additional information, namely the form of the data exchange between the device interface module and the user programs. For example, rather than provide barometric altitude directly, the device interface module might provide two or three quantities from which it could be computed. Such a design change would require a change in the function specification but not in the assumption list.

The two versions of the interface have different purposes: 1) the assumption list explicitly states assumptions that are implicit in the function specifications, making invalid assumptions easier to detect, and 2) the programming constructs can be used directly in user programs. It is essential that the two descriptions be consistent. The assumptions should be embodied clearly in the programming construct specifications, and the programming construct specifications should not imply any capabilities that are not stated in the assumption list. The assumption list should be reviewed by programmers, users, and hardware engineers who have the knowledge necessary to check it for validity and generality. For example, the A-7 assumptions were reviewed by system engineers and hardware engineers familiar with the A-7, A-6, and F-18 aircraft. Assumptions written in prose are easier for non-programmers to review. The specifications of the programming constructs should be reviewed by programmers who have worked with similar programs. These reviewers evaluate how well the abstract interface supports user programs and whether the device interface module can be implemented efficiently. The procedures and criteria used to review the A-7 device interface modules are described in section DI.iii.

Design Procedure

Obtaining a correct and consistent dual description of an abstract interface is an iterative process. Although we attempted to list assumptions first, many of the necessary assumptions were quite subtle and only became apparent when we worked on the design of the programming constructs. Review of the assumption lists revealed errors in the programming constructs. The interfaces in this document are the result of several cycles of internal review, as well as a formal review by the A-7 maintenance team at the Naval Weapons Center.

DESIGN PROBLEMS

The design considerations mentioned earlier serve as design guidelines and as standards for judging results. It is not always easy to apply them. Conflicts arise among three design goals: small device interface modules, device-independent user programs, and efficiency. What if user programs could use a device more efficiently if they could exploit assumptions that are not valid for all possible replacement devices? What if encapsulation of assumptions that are not always valid makes a device interface module slower or bigger? Acceptable compromises must be based on estimates of the likelihood of future changes. This section shows tradeoff problems and how we resolved them, attempting to minimize the expected cost of the software over its entire period of use.

Problem 1: Major variations among available devices

Deciding how much capability to include in a device interface module is particularly difficult when there are major differences among replacement devices. For example, new Inertial Measurement Set (IMS) models produce present position data; other IMS devices produce velocity data; and the current A-7 IMS produces only velocity increments. In order to simulate an IMS that produces present position using the current IMS hardware, much of the navigation software would have to be inside the IMS device interface module; the result would be a very large module. One must choose between (a) confining major changes within the device interface module, and (b) keeping the device interface module small, so that minor but more likely changes are easier to make. Our compromise limits the range of devices represented by our virtual IMS, with the understanding that the remaining differences can be confined to a small set of user programs. Although our virtual IMS does not provide present position, it does provide velocities rather than velocity increments; the velocity increments are only used to compute velocities, and the velocities are widely used. The resulting virtual IMS is considerably easier to use than the hardware IMS, yet we expect the IMS module to be reasonably small.

Problem 2: Devices with characteristics that change independently

Some devices have several sets of characteristics that can change independently. For example, the Projected Map Display Set (PMDS) consists of a set of filmstrips and a hardware drive that positions a filmstrip in a display. The same drive could be used with new filmstrips containing maps in a different format, and the same filmstrips could be used with a different drive. Two independent sets of characteristics should be hidden in different modules so that they can be changed independently. However it is unnecessary for user programs to be aware of the separation. We chose to hide both sets of characteristics in one PMDS device interface module. The module will later be divided into two submodules, each insulated from changes in the other. Since the division is a secret of the PMDS device module, it is not apparent to user programs and is not presented in the interface specification.

Problem 3: Virtual device characteristics that are likely to change

Some changeable device characteristics must be revealed to user programs so that they can exploit the device effectively. Examples include measurement resolutions, the number of positions on switches, and device limitations such as a maximum displayable value. Although we would like user programs to be insulated from all device changes, they must behave differently if such characteristics change. For example, user programs controlling the PMDS must behave differently if the virtual PMDS provides maps of three different scales instead of just two. We represent such characteristics by symbolic constants. Both user programs and device programs are written in terms of symbolic constants rather than actual values. At system generation time, code can be generated by conditional macro expansion based on the actual values of the parameters.

We defined system generation parameters for the range and resolution of input and output data because 1) range and resolution are highly likely to vary among different devices; and 2) this information is needed by user programs in order to perform arithmetic accurately and efficiently. User programs written in terms of system generation parameters do not need to be rewritten if the parameter values change.

Initially we assumed that all parameter values would be known at system generation time; i.e., we explicitly assumed that whenever a replacement device is introduced, a new version of the program will be generated and deployed with the device. This assumption was questioned at the design review. It is Navy policy not to have multiple versions of the software in the fleet, even though equipment changes cannot be made simultaneously. Furthermore, we cannot require a new system generation if a new device breaks down and is temporarily replaced by a device of the old type. As a result, some of the parameters must be changeable at run-time. In theory this is true for any of the parameters; in fact changeover problems are more likely for some devices than others. The cost of run-time variability also differs among devices, depending on whether the parameter can be used to control code generation so that run-time tests can be avoided. If changes are unlikely, we are reluctant to give up the efficiency advantages of binding the values at system generation time; if binding the value early causes no significant savings, we are reluctant to give up flexibility. We decided each case individually, using the following guidelines:

1) Parameters with a low cost for variability are treated as run-time variables, whatever the likelihood of change. Access functions to store and retrieve values appear in the module interface. See problem 4 for an additional problem about these parameters.

2) Parameters with a low likelihood of change and a high cost for variability are bound at system generation time.

3) For parameters with both a high likelihood of change and a high cost for variability, there are two possible solutions:

a) They can be treated as run-time variables, with the option to bind them earlier by providing values at system generation time. This option allows us to delay the final choice until we have more information.

b) We can find a conservative value that can be used for both devices, allowing us to bind the value at system generation time.

Problem 4: Device-dependent characteristics that vary at run-time

In some circumstances, user programs must handle device-dependent data. For example, when one IMS is replaced by another of the same type, the software must be adjusted because of manufacturing variations. The IMS software is parameterized so that it can be tailored to fit a particular piece of hardware. It is a requirement that we be able to change these parameters without reassembly. The parameters are entered at run-time through the computer panel. To receive the data, the IMS module provides access functions, which are called by the user programs that read in the panel data. Unfortunately, the existence of a run-time parameter such as drift rate reveals a secret of the IMS module, i.e., that the actual device drifts out of alignment. An additional drawback is that a replacement device might require different calibration data, requiring a change in the interface. We restrict use of these access functions so that the software making use of them is limited and easily identified. The restricted assumptions and functions are called reconfiguration interfaces and are appended to the normal interfaces.

Problem 5: Interconnections between virtual devices

Ideally, virtual devices would be independent of each other, allowing the associated abstract interfaces to be designed independently of each other. However the A-7 system has device interdependencies introduced for hardware convenience. Some of these interdependencies are based on assumptions made by hardware designers about the software. We can hide the nature but not the existence of the interconnections; if we hid the interconnections, later changes in the user programs might result in attempts to access the devices incorrectly. The existence of interconnections is revealed in the assumption lists in terms of restrictions on the use of virtual devices. For example, there may be an assumption that two virtual devices cannot be used simultaneously. If the hardware interconnection is later removed, additional uses of one or both devices may become possible and desirable. Making such additions will inevitably require changes on both sides of the abstract interface: changes in user programs to exploit the new capabilities and changes in device interface modules to remove the restrictions. Since this cannot be avoided, there is no loss in revealing the restriction.

A similar problem can arise within a single device interface module if the present hardware does not allow two capabilities of the virtual device to be used at once. Again, we chose to reveal the restriction to user programs even though it might not be true of future devices.

Interconnection problems also arise where one device (the provider) provides information used by another (the receiver). There are two cases to consider:

1) The computer can detect the failure of the provider. If so, the virtual receiver signals a failure when the provider fails, even if the actual receiver does not detect the failure. The device interface module for the receiver can simulate detection of the failure thereby hiding the interconnection.

2) The computer cannot detect the failure of the provider. The undetectable failure of the provider must be also considered an undetectable failure of the receiver. People writing user programs that rely on the virtual receiver must be aware that undetectable failures are possible, but they need not be aware of the interconnection.

Problem 6: Inconsistencies in the hardware interfaces

A hardware interface may provide similar functions in dissimilar ways. For example, the symbols on the HUD have three states: ON, OFF, and BLINKING. The HUD provides a hardware blink command for some symbols, but for others the software must simulate the BLINKING state by alternating the symbol between the ON and OFF states. Whenever the hardware interface provides some means to perform the action, we provide the feature consistently in the virtual device. The virtual HUD has commands to blink all symbols. However, when the hardware interface does not provide a way to perform an action, we were forced to reveal the inconsistency in the interface. For example, some HUD symbols have only two states: OFF and BLINKING. Since there is no way to simulate the ON state, the virtual device cannot provide it. If the hardware limitation is removed in the future, the inconsistency can be removed from the virtual device. It is unavoidable that this change would require changes on both sides of the abstract interface: changes in user programs to exploit the new capability and changes in the device interface module to implement it.

Problem 7: Switch nomenclatures

Many of the switches do nothing more than set a bit that the computer can read. The label on a switch is an easily changed characteristic and could be hidden. We could name the switches anonymously (e.g. with integers) and use non-mnemonic names for the settings. However, a change in switch nomenclature will most likely be accompanied by a change in the requirements. The expected cost of working with non-mnemonic names, i.e., more errors, is high. In line with our basic principle of trying to minimize the expected cost of the software over its whole period of use, we have chosen to reveal the nomenclature in switch names and mnemonic values for the switch settings. The names may suggest more tnan is actually stated in the assumptions; programmers are cautioned not to make assumptions about the switches beyond those that are explicitly stated in the interface documents.

Problem 8: Switches with hardware side-effects

When a switch also affects other devices, the meaning associated with it is not solely a software decision; major hardware changes would be required to use it for any other purpose. We consider such a switch part of the device that it affects, even if it is not physically located with the device. As far as user programs are concerned, the switch does not exist; the effects of the switch appear as changes in the operating mode of the virtual device. For example, the "Terrain Following" switch located on the master function panel affects the state of the Forward Looking Radar (FLR). Instead of appearing in the same interface description as the other master function switches, it is hidden within the FLR device interface module. User programs cannot read the switch, but they can call an access function that reveals the operating mode of the FLR.

Problem 9: Reporting changes in device state

User programs are often required to respond quickly to a change in device state. For example, user programs determining the current navigation mode need to know when the reliability status of the IMS sensor changes. The device interface module can either 1) provide an access function reporting the current state of the device, or 2) signal the state-change event. The choice of a mechanism depends on whether user programs base decisions on the current state or wait for the state to change. If we based the design on the requirements of user programs, changes in their requirements might result in changes in the device interface module, violating the design goal that the device module should not need to change unless the device is changed. We chose to specify both mechanisms in every case, but plan to implement only the ones that are actually needed. Thus requirements changes may require changes to device interface modules, but these changes will consist of implementing previously specified features. By specifying possible additions in advance, we expect to reduce the cost of later reprogramming.

Problem 10: Devices requiring information from the software

Some hardware devices require information that is not calculated within the associated device interface module. For example, the current IMS device needs to know whether or not the aircraft is above 70^o latitude, even though latitude is not calculated within the IMS module. One must choose between two ways to get the information to the device: 1) the device module can provide an access function that a user program calls in order to provide the information; or 2) programs in the device module can call other programs to get the information. Our decision is based on whether or not the information requirement is common to the class of replacement devices. If it is, the device interface module provides access functions for receiving the information. This solution results in added requirements for the rest of the software, and the user programs supplying the information must change if the information need changes. If the information need is peculiar to a particular hardware device, device interface programs call other programs to get the data. As long as the needed information is available from the rest of the system, no program outside the device interface module needs to change if the need for information disappears. We chose the latter solution for the IMS example because not all IMS devices require a signal from the computer at 70^o latitude.

Problem 11: Virtual devices that do not correspond to hardware devices

Initially, we assumed that there would be one virtual device for each hardware device. We found that modeling the virtual devices on the actual devices does not always result in clear interfaces. Some related capabilities are scattered among several hardware devices; some unrelated capabilities occur in the same device for physical convenience; other groupings can only be explained in terms of historical development. For example, weapons-related capabilities in the A-7 are scattered among several devices. Some weapons data comes from the device that controls weapon release, some is stored in tables, and some is provided by the pilot through switches. Additionally, the weapons release device fills two distinct roles: it is both a source of input data and the device that releases a weapon under computer control. Our final design includes one virtual device for weapons release and one for weapon data. The virtual devices are much simpler to understand than the actual devices. It is important not to be unduly influenced by the physical location of hardware units.

Problem 12: Device tests

The software is required to perform device tests by transmitting specified values, and in some cases reading back a check value. The relationship between the output values and the input value is clearly device dependent. Some of the tests affect displays that can be seen by the pilot. One can argue either that the values used for testing are user-specified requirements or that they are device characteristics. If we place the choice of test values outside the device interface modules and the test procedures inside, the interface will be complex and probably reveal some device characteristics. If we hide the test values within the device interface modules, a change in the test policies might require module revision even though the device was not changed. We chose the latter alternative, assuming that the choice of test values is sufficiently tied to device characteristics that the values would change if and only if the device changed. Including the test values within the device interface modules results in a simple interface that is easy to use. On the other hand, if the values were chosen in order to be easily checked by a pilot, our assumption is incorrect and the decision is wrong.

Problem 13: Choosing the correct access function to fulfill each requirement

The software requirements in reference (1) are expressed in terms of hardware data items. Since the existence of these data items is a secret of the device interface module, they are not mentioned in the device interface module specifications. Consequently a programmer will have difficulty choosing the right access function to implement each function. This problem would have been avoided had the access function specifications been available when the software requirements were written; the requirements could have been expressed in terms of effects on the virtual devices rather than the actual devices. We could rewrite the software requirements in terms of the virtual devices. However the payoff would probably not be worth the effort. Programmers could look at the implementations of the device interface modules in order to find the access function associated with a particular hardware data item. However this would be a clear violation of the information-hiding principle (ref (6)): extra dependencies between program components might be added because programmers could make assumptions about devices that were not explicitly stated in the abstract interface. As a compromise we added a section giving the access functions that the programmers must use in order to affect the data items mentioned in the software requirements. Programmers of user programs may assume that a certain requirement can be fulfilled by calling a certain access function; that assumption should continue to be valid even after a device is replaced and the device interface module is reimplemented.

The mapping is given in appendix 5, separate from the rest of the device interface specifications because it is only valid for requirements that were expressed in terms of the current actual devices.

Problem 14: Information required for diagnostic displays

The software requirements (ref (1)) require the ability to display raw input values exactly as they are received from the hardware devices. This information is used by maintenance personnel for diagnostic purposes, and will not be used within the software for any other purpose. If we included commands to read data directly in a restricted section of each interface, we would be revealing the association of virtual devices with hardware data items, an association that these modules are supposed to hide. We chose to allow the user programs that control diagnostic displays to bypass the device interface modules and use the data items directly. The disadvantage of this approach is that it is possible that a change in the input hardware will affect two modules: one providing a virtual device and one controlling a diagnostic display. This will not necessarily occur. The diagnostic display programs are not affected by changes in the meanings, ranges, bit encodings, or measurement errors associated with the input data; they need only be changed if a data item is removed or added, or if the number of bits in a raw value changes.

Problem 15: Devices tailored to system functions

The interface to the computer panel was among the most difficult to design. The hardware device is complicated because it is tailored to the system functions and contains a number of seemingly arbitrary features. For example, the display component consists of upper, lower, and side windows with 7, 6, and 1 seven-segment symbols respectively. Data can be displayed in different formats in the upper and lower windows by selectively turning on or off the lights that appear between some, but not all, of the display symbols. It is very different from the general purpose terminals. The procedure for pilot input of data is also complex and appears to be device-dependent. Originally we designed a virtual panel that hid all knowledge of windows and display formats. We identified all of the types of data that could be communicated through the panel and provided a separate input and/or output access function for each.

When we tried to use the panel, we repeatedly discovered situations in which the program is required to use the panel in unexpected ways. Since we were forced to add new functions for special cases, the interface underwent numerous changes and kept growing more complex.

We found that we were hiding two kinds of secrets in one module. In addition to the device characteristics, we had hidden aspects of the requirements. We also realized that other aircraft computer panels look very much like the present A-7 panel. This can be explained by the highly specialized application area and the inability of a pilot flying an aircraft to exploit the flexibility offered by a general purpose terminal. While it is theoretically possibile for the A-7 panel to be replaced by a very different display and data entry device, the likely replacement devices have most of the features that we were attempting to hide.

Our present design hides only those characteristics of the panel that seem truly arbitrary, such as the bit encoding of the various symbols. Characteristics such as the formatting lights and the number of symbols in the windows are now visible. Anything that could be done with the hardware can be done with the device interface module, but it can be done more easily.

The original design has not been discarded. It will form the interface to another module called the panel procedures module, which will provide common support services to the programs that use the panel. The panel procedures module will hide formats and the pilot communication procedures. Changes associated with replacement or alteration of the panel device can be confined to the panel device interface module only if a) the new panel offers essentially the same capabilities as the old, and b) there are no associated changes in pilot procedures. Both modules could be affected by a radical change, such as replacing the panel with a glass teletype, if the added capabilities of the new device are to be exploited. Only the panel procedures module would be affected by a change in the procedures used by the pilot to input data.

The lessons to be learned by this are that (a) not everything about a device that appears complex and arbitrary should be hidden in a device interface module, and (b) if you have to look at the requirements to design (in contrast to check) the device interface, you are probably making a basic error. Here again we see that if two secrets can change independently, they should be in separate modules.

DI.ii STANDARD ORGANIZATION FOR MODULE INTERFACE DESCRIPTIONS

Purpose

All interface specifications for device interface modules follow a standard organization. This section presents the organization, listing the subsections that appear in each device interface module specification and describing for each subsection the contents, format, and notation.

TABLE OF CONTENTS

SUBSECTION DESCRIPTIONS

Subsection 1: Introduction.

Subsection 1 describes the virtual device provided by the module. First it characterizes the device as one of the following: a sensor, a data entry device, a display device, or a control device. Sensors detect the state of the aircraft relative to the physical world; they are characterized in terms of the physical quantities they measure or the physical facts they detect. Data entry devices receive information from the pilot; these devices are characterized by the kind of information the pilot can enter, e.g. strings of numbers or switch settings. Display devices convey information to the pilot; these devices are described in terms of the manner of communication, e.g. whether visibly or audibly, whether as a string of numbers, as a position on a dial, or as a symbol positioned in a plane. Control devices generate signals to other devices in the avionics system; these devices are described in terms of the interpretations placed on the signals by the other devices.

Subsection 1 serves as a brief digest of the information in the rest of the device interface module specification. It also appears as an abstract for the virtual device in the table of contents for the report.

Subsection 2: Interface Overview.

Subsection 2 is composed of ready-reference tables allowing a reader to see at a glance the facilities provided by the module. Maintainers familiar with the module interfaces can use them to refresh their memories about particular facts without having to reread the explanations contained in later subsections.

Subsection 2.1: Access Function Table. The access function table contains an entry for each access function provided by the module; each entry includes function name, parameter data, and applicability conditions. Applicability conditions are expressed in terms of undesired events (see also subsection 7). More information about the functions is provided in subsections 5 through 8. The table follows the format specified in figure 1 and illustrated in figure 2. For the sake of brevity, the table entries are organized into groups that have the same potential undesired events. These groups are separated in the table by horizontal lines.

Subsection 2.2: Events. The event table contains a list of all of the events signalled by the module. Events are denoted by the @T(Condition)/@F(Condition) notation defined in reference (1). "Condition" must be an entry in the local dictionary (see subsection 6). Event meanings must be easily inferred from the associated dictionary entries.

Subsection 2.3: Virtual device modes. Some virtual devices have different operating modes affecting the facilities provided by the device interface module at a given time. This table lists the modes, each denoted by !*mode name*!, and lists the access functions that may be called legally in each mode. This table does not show how the current mode is determined: mode changes caused by user program actions are described under function effects (see subsection 8); the access function table shows how to detect mode changes that occur within the module.

Figure 1: Function Table Format

```
Function name                Parm type        Parm info          UEs

+function1+                  p1:type1;K       info1              %name1%
                             p2:type2;K       info2              %name2%
                                  .               .
                                  .               .                 .
                             pN1:typeN1;K     infoN1                .
                                                                 %nameM%
+function2+                  p1:type1;K       info1
                             p2:type2;K       info2
                                  .               .
                                  .               .
                             pN2:typeN2;K     infoN2

     .
     .
+functionG+                  p1:type1;K       info1
                             p2:type2;K       info2
                                  .               .
                                  .               .
                                  .               .
                             pN2:typeNG;K     infoNG
```

LEGEND

Underscored symbols are required (without the underscores). Other names and letters are defined as follows:

G	:	number of functions in the group, where group is defined as a set of functions with the same entries in the UE column
functionJ	:	name of the Jth function in the group, where J = 1,...,G
NJ	:	number of parameters for the Jth function. If zero, the parameter columns are empty for the function.
pL	:	the Lth parameter of a function, where L = 1,...,NJ
typeL	:	type of parameter pL: either a system-defined type (figure 3), or a type defined in the local type dictionary (see subsection 5)
K	:	I, O, or IO for input, output, and input-output parameter, respectively. Functions receive the values of input parameters and deliver the values of output parameters. Input-output parameters serve both purposes.
infoL	:	definition of the meaning of parameter pL; may be an entry in the local dictionary (!+entry L+!) or an expression involving other parameters, such as "p1 + p2"; infoL may be omitted for any parameter whose meaning is given in the effects subsection (8), or it may optionally be a catch phrase summarizing the function effect description.
M	:	number of UE dictionary entries defined for the group
nameE:	:	name of a UE dictionary entry E, where E = 1, 2, . . M; the dictionary entry defines the circumstances that cause a function call to be illegal

Figure 2: Example Access Function Table
(Extracted from DI.9)

Function name	Parm type	Parm info	Undesired events
+G_IMS_MAG_HEADING+	pl:angle;O	!+heading MAG+!	none
+G_IMS_MAG_VARIATION+	pl:angle;O	!+magvar IMS+!	
+G_IMS_MODE+	pl:imsmode;O	!+IMS mode+!	
+G_IMS_STATUS+	pl:logical;O	!+IMS ready+!	
	p2:logical;O	!+IMS rel+!	
+S_IMS_ENABLE+	pl:logical;I		
+S_IMS_SCALE+	pl:imsscale;I		
+G_IMS_PITCH+	pl:angle;O	!+pitch IMS+!	%Coarse rotation in progress%
+G_IMS_ROLL+	pl:angle;O	!+roll IMS+!	
+G_IMS_TRUE_HEADING+	pl:angle;O	!+heading IMS+!	
+G_COARSE_ROTATING+	pl:logical;O	!+IMS rotating+!	%IMS disabled%
+S_X_COARSE_ROTATION+	pl:angle;I		
+S_Y_COARSE_ROTATION+	pl:angle;I		
+S_Z_COARSE_ROTATION+	pl:angle;I		
+G_IMS_E_VELOCITY+	pl:speed;O	!+E vel IMS+!	%Vel not init%
+G_IMS_N_VELOCITY+	pl:speed;O	!+N vel IMS+!	%IMS disabled%
+G_IMS_V_VELOCITY+	pl:speed;O	!+V vel IMS+!	%Coarse rotation in progress%
+S_IMS_E_VELOCITY+	pl:speed;I		%Coarse rotation in progress%
+S_IMS_N_VELOCITY+	pl:speed;I		%IMS disabled%
+S_IMS_V_VELOCITY+	pl:speed;I		
+S_X_FINE_ROTATION+	pl:angle;I		
+S_Y_FINE_ROTATION+	pl:angle;I		
+S_Z_FINE_ROTATION+	pl:angle;I		

Subsection 3: Assumptions Lists.

Subsection 3 lists assumptions that are expected to remain true even if the actual device is replaced by another device. These assumptions should characterize in detail the virtual device and should only contain facts that are sufficiently essential and stable that programmers using the interface may depend on them not to change. The purpose of the assumption list is to serve as an explicit medium for review by non-programmers as well as programmers. The assumptions list is presented in two sections.

Subsection 3.1: Basic Assumptions.

This subsection lists the basic assumptions about the normal use and operation of the device. The assumptions may concern any of the following:

a) information available from the module and correlations between the information available,

b) information that must be supplied to the module in order for it to operate correctly,

c) events reported by the module,

d) operating states of the module and how they affect the information available and the information required,

e) failure states of the module and how they affect the information available, and

f) suggestions about how to program efficiently using the facilities provided by the module.

Subsection 3.2: Assumptions about Undesired Events.

This subsection lists assumptions made about undesired events For example, a device may assume that it will never be accessed if the aircraft is not airborne. In the development version of the system, the undesired event listed in the function table will be signalled if this assumption is violated by a user program. In the production version of the system, the undesired event handling code will be removed, so that results are unpredictable. There should be an assumption in this subsection for each undesired event listed in the access function table.

Subsection 4: Interface Design Issues.

If any alternative assumptions, functions, or function parameters were considered, they should be briefly described and the reason for the decision given. In the early design stages, this subsection serves as a history of design decisions, so that the same issues are not rehashed over and over again. In the final document, the subsection will serve as a design rationale and as guidance to maintenance programmers revising the program. This subsection should not discuss implementation issues.

Overview of subsections 5 through 9

Subsections 5 through 9 supplement the access function table. Subsection 5 defines types that are used only for this interface; system-defined types are given in figure 3. Subsection 6 defines !+terms+! used in the tables. Undesired events are denoted by %entry% and defined in subsection 7. Subsection 8 describes the effects of function calls on the operating state of the virtual device. Subsection 9 gives the characteristics of the virtual device that can be changed at system generation time without any reprogramming.

The subsection descriptions are preceded by descriptions of access function types and of system defined types.

Function Types

Listed below are four types of access functions. Each type is characterized by the facilities that access functions of that type offer to user programs, their effects on the other access functions provided by the module, information required to specify them, and naming conventions.

Value Functions. Value functions deliver values to user programs by means of output parameters. A call to a value function has no effect on subsequent calls to that function or any other function. Whenever value functions are specified completely by entries in the access function table, there are no associated entries in the effects subsection. Value function names begin with "G_" for "Get value". For an example of a value function, see "G_ADC_BARO ALTITUDE" in section DI.1; this access function returns the most recently measured barometric altitude.

Effect functions. Effect functions enable user programs to affect the future operation of the module by passing it information or giving it commands. Effect functions may affect the values returned by value functions; they may change the values shown by display devices; or they may affect the current operating mode of the device interface module. These functions do not return values themselves. The parameter-information column in the access function table can be left blank for these functions whenever the function effects subsection adequately defines the parameter meanings.

The names of effect functions usually begin with "S_" for "Set value." For examples of effect functions, see +START_DRS+ in section DI.5 and +S_IMS_N_VELOCITY+ in section DI.9. The former function turns on the virtual doppler device, so that the other access functions provided by the module may be called legally. The latter function initializes the velocities maintained by the inertial measurement set device interface module.

Matching Value/Effect Function Pairs. In some cases, value and effect functions are matched so that the value function always returns the most recent value passed in by the effect function, and the effect function has no other effect. For the sake of brevity, these functions are described together since they share semantics. Matching function pairs always share the same name except that the value function starts with "G_" and the effect function starts with "S_". For an example, see +G_ADC_LPROBE+ and +S_ADC_LPROBE+ in section DI.1R.

Hybrid functions. Hybrid functions have characteristics of both value and effect functions: they return values and affect the future operation of the module. These functions will usually be described by both parameter-information entries in the access function table and descriptions in the effects subsection. There are no naming conventions for these functions, but the names should be as meaningful as possible. For an example of a hybrid function, see +TEST_ADC+ in section DI.1. This function causes the ADC device to perform its built-in self-test; it also returns the result of the test.

System-defined types

Figure 3 gives the system data types, that is, data types that are so widely used in the device interface specifications that they are defined here for the whole report. These types need not be defined in the local type subsections. The Extended Computer module in the A-7 system will provide access functions to declare variables of system types, to manipulate values, and to convert results from one type to another. The system types can be divided into two categories: simple types, which have no units, and specialized types, which are designed for measurements that can be given in various units. A value of a specialized type has no units; to associate units with it, a programmer must call an access function to convert it to a real value, specifying the units he wants. Thus a single value of type angle can be converted to different values of type real, one representing degrees, one representing radians, and one representing circles.

Figure 3 shows whether a type is classified as simple or specialized; for specialized types, it gives the units into which the values may be converted. For more information, see documentation for the Extended Computer module (reference (5)).

Figure 3: System Data Types

Type	Type class	Possible Units after Conversion
angle	specialized	circles, degrees, radians, sine/cosine, octant/magnitude
bitstring	simple	
char	simple	single characters
distance	specialized	feet, nmi
integer	simple	special case of real, with integer resolution
latitude	specialized	special case of angle, with range (in degrees) limited to ± 90
logical	simple	possible values: true/false
longitude	specialized	special case of angle, with range (in degrees) ± 180
mach	specialized	mach number
pressure	specialized	inches of mercury
real	simple	
speed	specialized	feet per second, feet per minute, knots
time	specialized	milliseconds, seconds

Subsection 5: Local Types

Subsection 5 defines local data types, that is, data types that are not of general interest for the rest of the A-7 system. Most of these types are enumerated types, that is, the type is defined by a list of acceptable values. Figure 4 demonstrates the format used for defining local types.

Figure 4: Local Type Dictionary Format

type name	type definition
imsmode	enumerated: $Gndal$, $Norm$, $Iner$, $Magsl$, $Grid$, Off
weap_type	integers: 0, 1, . . . NUM_WEAP_TYPES

Subsection 6. Local Dictionary

Subsection 6 defines terms denoted by !+term+! that are used to specify meanings for access function parameters and for events. Figure 5 demonstrates the format used for defining terms.

Figure 5: Local Dictionary Format

!+IMS ready+!	True iff IMS is ready for operation under computer control
!+IMS rel+!	True iff IMS is reliable, according to its own self-test
!+Nvel, IMS+!	The velocity of the aircraft measured along the Y axis of the IMS. When the IMS is properly aligned, this measures north/south velocity of the aircraft.

Subsection 7: Undesired Event Dictionary

An undesired event (UE) occurs when an access function is called incorrectly, either with an incorrect parameter or at a time when it is illegal because of the current operating state of the module.

An undesired event will be signalled when any access function is called with a parameter that is the wrong type or out of the specified value range. Since these undesired events are universal, they are not defined individually for each function.

An operating state UE is considered enabled when the UE may occur, and inhibited when it cannot occur. If a UE is enabled, it will occur when an access function associated with it is called; the group of associated access functions are given in the access function table. The operating state of the module, and therefore the enabled/inhibited status of UEs, can be affected either by user commands or changes detected within the module. For example, the inertial platform module (IMS) is affected both by user commands (+S_IMS_ENABLE+) and by changes in the actual IMS hardware (whether the platform is ready and reliable).

Subsection 7 defines the %term% entries in the access function table. The specifications describe user-controlled state UEs in terms of the commands that inhibit or enable them, and internal state UEs in terms of the value functions that reveal whether the UE is currently inhibited or enabled.

The specifications should refer to function parameters in terms of the "p" notation used in the access function table in Subsection 2.1. Figure 6 demonstrates the format used in this section.

Figure 6: Undesired Event Entries

%IMS disabled%	the IMS is not enabled for computer control. <u>enabled</u> when @T(!!power up!!) affected by +S_IMS_ENABLE+: <u>enabled</u> when p1 = <u>false</u>; <u>inhibited</u> when p1 = <u>true</u>
%DRS not in Normal%	Occurs if DRS not in !*Normal*! mode when a function is called that is only legal in !*Normal*! Revealed by +G_DRS_MODE+ p1 = $Normal$ shows inhibited; otherwise enabled
%Sqrt of negative no%	squareroot function called for a negative number
%Vel not init%	Initialized: <u>enabled</u> when @T(!!power up!!) <u>enabled</u> by +S_IMS_ENABLE+ with p1=<u>false</u> <u>inhibited</u> by associated +S_IMS_?_VELOCITY+, where ? = N, E, or V

Subsection 8: Function Effects

Subsection 8 provides informal specifications for effect and and hybrid functions. These specifications are expressed in terms of 1) which value functions are affected and how, 2) which undesired events are inhibited or enabled, and 3) any restrictions affecting the occurrence of enabled UEs. The function specifications should refer to function parameters in terms of the "p" notation used in the access function table in Subsection 2.1. The functions should be grouped in any way that facilitates a concise presentation. Figure 7 demonstrates the format used in subsection 8.

Figure 7: Example of an Effect Specification

+S_IMS_N_VELOCITY+ +S_IMS_E_VELOCITY+ +S_IMS_V_VELOCITY+	These functions initialize the IMS velocities and inhibit %Vel not init% UEs. Future values returned by +G_IMS_?_VELOCITY+ will be based on p1 and the velocity changes that have been measured since +S_IMS_?_VELOCITY+ was most recently called.
+S_IMS_ENABLE+	This function enables the IMS device for computer control. When called with p1 = true, %IMS disabled% is inhibited. When called with p1=false, %IMS disabled% and %vel not init% are enabled; %Coarse rotation in progress% is inhibited, coarse rotations are stopped, and accumulated coarse rotation requests are lost.

Subsection 9: System Generation Parameters

Subsection 9 lists the characteristics of the module that can be changed by setting parameters at system generation time. These parameters are denoted by ¢parameter¢, and may be used as symbolic constants by users of the interface. Figure 8 demonstrates the format of the system generation section.

Figure 8: System Generation Parameter Specification

¢IMS_ATTITUDE_RES¢	Resolution of IMS attitude measurements
¢NUM_MAP_SCALES¢	Number of map scales available

Subsection 10: Information Hidden.

Subsection 10 lists secrets of the module. The goal of this list is to help maintenance programmers find the module where a particular change should be made. This subsection serves as the complement of the assumptions sections: it lists the aspects of the module that are expected to vary between different implementations of the module. In the early design stages, the two subsections should be reviewed together; reviewers should be confident that every important feature of the device appears in exactly one of the two lists. Misunderstandings about module responsibilities should be revealed by reviews of this subsection.

This subsection should characterize only the nature of the secrets, and should avoid giving away any details that should not be assumed by users of the module. Figure 9 demonstrates the format of this section.

Figure 9: Example Module Secrets

1. The format of data received from the sensor, including the scale and offset.
2. Corrections that must be applied to raw sensor readings to get accurate measurements.
3. Methods used to determine if the sensor readings are valid.

Subsection 11: Calculations Performed Within the Module.

Subsection 11 lists calculations that are performed within the module and states why they belong in this module. Acceptable reasons include that the calculation is intended to be a secret of the module, that the calculation requires hidden information (e.g., a device dependent calculation), that the calculation is more efficiently performed within the module, or that the calculation might be performed by hardware in some replacement devices. The purpose of the section is to clarify what is and is not done within the module, in order to help maintenance programmers find the proper place to make a change.

Subsection 12: Implementation Notes.

During the design of the module interface, certain facts or ideas may come to the designer, ideas that would be necessary or useful to future implementors. These should be noted in this subsection so that they are not lost. The subsection will be removed from the final interface documentation and included in the module implementation description.

DI.iii REVIEW INSTRUCTIONS FOR ABSTRACT INTERFACES

BACKGROUND

The abstract interface specifications in this document were reviewed by outside but interested parties, i.e., hardware, software, and systems engineers for the A-7, A-6, and F-18 programs at the Naval Weapons Center. In order to have the design scrutinized as thoroughly as possible, we designed the review procedures documented in this section. We expect to use the same procedures to review the document again whenever we change it substantially.

Effective reviews require collaboration between designers and reviewers. Designers may be very familiar with their design products, but they lack objectivity. Outside reviewers have a fresh perspective, but they may have to devote so much mental energy to understanding a novel design that they are unable to review it critically. If a design is unconventional, they may not know what properties it should have or what kind of errors to look for. To help the reviewers work methodically, we prepared a list of questions for them to answer as they studied the design documentation. During the design reviews, at least one designer met with each reviewer to compare answers to questions and resolve discrepancies. This procedure caused the outside reviewers to take a more active role in the design review than they usually do.

We did not find it necessary for every reviewer to review the entire product nor to answer the same questions. Often a design needs to be reviewed from several different viewpoints, by people with different expertise. In the review instructions, we grouped the questions into five different categories and assigned them to reviewers with different backgrounds. While this highly structured review process was helpful, we saw the danger that we could distract the reviewers from problems that we had overlooked by making them focus on problems we had anticipated. To reduce this danger, we assigned device interface specification to an additional reviewer for a general, unguided review.

The rest of this section describes four different types of reviewers, discusses the goals of the five review categories, and lists the questions belonging to each category.

For the convenience of the reviewers, the questions were arranged in questionnaires to be filled out either while they read the document or while they discussed it with the designers. By providing questionnaires, we ensured that the same questions were asked about all parts of the design. We also find the filled-in questionnaires a useful record of the comments made during the review. We have included blank copies of the forms at the end of this section.

REVIEWERS

We want each specification reviewed from the four different points of view described below. Although we need at least four different people to review each device interface module, no one person will be asked to review <u>every</u> module, and most reviewers of a particular module need not read the entire specification. The reviewers need not all be from the A-7 project; nor do they all need to be programmers.

<u>Point of view</u>	<u>Expertise Required</u>
DEVICE EXPERT	Familiarity with the device used on the A-7 and with similar devices found on other aircraft. These people ought to know about several devices of this type, about the technology used to build such devices, and about past changes and future trends in these devices. They need not be programmers; people involved with the design or purchase of such devices would be fine.
DEVICE PROGRAMMER	Experience writing or modifying programs that deal with this device or others of the same type. They should be familiar with tricks to use the device effectively with minimum consumption of computing resources. If people with experience on the A-7 cannot be found, people who have used similar devices for other aircraft are acceptable. There are two separate tasks to be performed from this viewpoint, and it is better if they are carried out by different people.
AVIONICS PROGRAMMER	Good logical minds and familiarity with avionics programming in general. These people need not have experience writing programs for these specific devices. They will be asked to perform certain checks for <u>internal</u> consistency and completeness; consequently a lack of information on the specific devices may be advantageous.
A-7 REQ EXPERT	Sufficient familiarity with our A-7 software requirements document (ref (1)) to be able to read the function descriptions (section 4) that refer to the devices hidden in this module. Familiarity with the A-7 application beyond this document is not essential. We value the ability to make disciplined logical completeness checks above familiarity with the A-7. However, they should have experience writing programs similar to the A-7 software.

REVIEW CATEGORIES AND QUESTIONS

For each review category, we give the appropriate reviewer viewpoint, a reading assignment, and a list of specific questions. The reading assignments are given in terms of subsections in the device interface specifications.

Review A: Assumption validity

All assumptions should be valid for any device that can reasonably be expected to replace the current device. DEVICE EXPERT reviewers should review the specifications for this criterion. They should read the assumptions (subsection 3) and answer the following questions for each assumption:

A1: Is this assumption valid for the current device? If not, explain.

A2: Is this assumption valid for possible replacement devices, including

- devices already on the market,
- devices under development, and
- devices for which a need has been expressed?

If not, explain.

A3: Is this assumption necessarily valid for all devices of this type? Explain why or why not.

For the entire list of assumptions for a particular device, they should answer:

A4: Are there additional assumptions that can safely be made about the device, i.e. assumptions for which the answers to questions A1 - A3 would be "yes."

Review B: Assumption Sufficiency

The assumption list should contain all the assumptions needed by the user programs in order to make effective and efficient use of the device. The device interfaces should be reviewed for this criterion by DEVICE PROGRAMMERS. They should refamiliarize themselves with device-dependent programs they have written or maintained, read the assumptions (subsection 3), and answer the following questions:

B1 What additional information do you need in order to design efficient algorithms to use this device? Do not mention detailed facts such as bit-level device register formats, but more basic qualities of the device such as capabilities, costs of certain operations, or operating constraints.

B2 Consider the programming techniques, algorithms, methods, tricks, etc. used in writing programs that interact with this type of device. What techniques, algorithms, methods, etc. have you used that require further knowledge of the device than is contained in the assumptions?

Review C: Consistency between Assumptions and Functions

The assumptions should be compared to the function and event descriptions to detect whether (a) they are consistent, and (b) the assumptions contain enough information to ensure that the functions can be implemented and the events can be detected. If an access function cannot be implemented unless the device has properties that are not in the assumption list, there is a design error, i.e., either a gap in the assumption list or a function that cannot be implemented for some replacement device. The device interface specifications should be reviewed for this criterion by AVIONICS PROGRAMMER reviewers. After studying the assumptions (subsection 3), the design issues (subsection 4), and the functions and events (subsections 2 through 8), they should perform the following reviews:

Review C-1: For each of the access functions the reviewer should answer the following questions:

C1 Which assumptions tell you that this function can be implemented as described?

C2 Under what conditions may this function be applied? Which assumptions describe those conditions?

C3 Is the behavior of this function, i.e., its effects on other functions, described in the assumptions?

Review C-2: For each of the signalled events the reviewer should answer the following questions:

C4 Which assumptions tell you that this event can be detected by the virtual device?

C5 Are there limitations on when the event can be detected? Which of the assumptions tell you these limitations?

Review C-3: For each of the undesired events, the reviewer should answer the following questions:

C6 Which assumptions tell you that this undesired event can occur and that it can be detected?

C7 Is the undesired event described in terms that are meaningful to a user of the virtual device, i.e. does the description refer only to aspects of the device that are revealed in the assumptions, function effects, dictionary, or local data type definitions? If not, what aspects of the actual device are revealed?

Review C-4: For each of the local data types (section 5), the reviewer should answer the following questions:

C8 What is the range of values for variables of this type?

C9 Which assumptions tell you the possible values for this variable?

Review C-5: For each of the system generation parameters (section 9) the reviewer should answer the following questions:

C10 What is the range of values for this parameter?

C11 Which assumptions tell you the possible values for this parameter?

Review C-6: For each of the terms in the local dictionary (section 6), the reviewer should answer the following questions:

C12 Is the definition clear?

C13 Is there any confusion between this and other terms that you have seen defined or used in this document?

C14 Is this term defined in a way that is meaningful to the user of the virtual device or is it defined in terms of the hardware?

Review C-7: The reviewer should also answer the following questions about the entire reading assignment:

C15 Which assumptions were not used in answering questions C1 - C14?

C16 Which assumptions contain information that was not used in answering questions C1 - C14?

C17 Which assumptions contradict each other?

C18 Are the interface design issues clearly stated and the decisions well-motivated?

Review D: Access function adequacy

User programs should be able to use the device efficiently using only the access functions provided in the abstract interface. The specifications should be reviewed for this criterion by DEVICE PROGRAMMER reviewers. For a given specification, it is better if different people perform reviews B and D. They should read the access function information (subsections 2, 6, 7 and 8), consider programs they have written or maintained that interact with this device, and answer the following questions:

D1 Is there anything that you do in your present programs that you could not do if you had to rewrite them in terms of calls on these functions?

D2 Are there places where a rewritten program would use more space than your present program does? Please explain.

D3 Are there places where your present program is faster than a rewritten program would be? Please explain.

D4 Are there any situations in which your rewritten program would call an access function that performed more actions than were actually needed? Please give examples.

D5 Are there any tricks that you use when programming for this device that could not be used if you used the functions from this module?

D6 Are there any additional functions that would be useful?

D7 Are there any functions that would be better split into two or more separate functions?

D8 Are there any pairs of functions that are (a) always called together and (b) that could be implemented more efficiently if combined into a single function?

D9 Would you prefer the virtual device to the real one? Why or why not?

D10 Are there any errors detected by your program not reported by these functions?

D11 Are there any errors detected by these functions not detected by your program?

Review E: Adequacy of the Data-Item/Function Index

Programmers must be able to figure out how to meet the software requirements associated with devices without referring to the module implementations. Since virtual devices hide the actual devices, the specifications may not show how to write programs to meet requirements expressed in terms of the actual devices. For this reason, the device interface module specifications include a section intended to guide the programmer working from the software requirements specification to the correct access function for his purposes. A-7 REQUIREMENTS EXPERT reviewers should review the data-item/function index for effectiveness. This review can be conducted on a function-by-function basis working from the A-7 software requirements document (ref (1)). For each function in section four of the requirements, reviewers should answer the following questions:

E1 Does the index give you enough information to write programs that meet the requirements as described in the requirements document? What additional information is needed?

E2 Are there any situations where, by following the instructions given in the index to perform some operation, you will cause some unwanted or unnecessary side-effects? Please explain.

E3 Are there any situations where, by following the instructions given in the index, you will cause the device interface module to do more work than is actually necessary? Please explain.

Questionnaire for Review A: Assumption Validity

Reviewer ______________ Date _________

Interface ______________

Insert a brief (4 words or less) keyword description in the following table for each assumption belonging to the interface. Then fill in each entry in the table with a Y (Yes) or N (No) to answer each question for each assumption. The rows of the table are labelled according to assumption, the columns according to question. As an example, an entry of Y in row 2 column 3 means an answer of Yes to question A3 for assumption 2. For each N answer, use an explanation sheet to give the answer requested in the question. Space to answer question A4, which applies to all assumptions, is provided below the table.

	Question		
Assumption	A1	A2	A3
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			

A4 (Additional Valid Assumptions):

Explanation Sheet for Review A Questionnaire

Reviewer ______________

Interface ______________

Explanations

Question/Assumption

1. ____/____

2. ____/____

3. ____/____

4. ____/____

5. ____/____

6. ____/____

7. ____/____

8. ____/____

9. ____/____

10. ____/____

Questionnaire for Review B: Assumption Sufficiency

Reviewer ______________ Date __________

Interface ______________

B1 What additional information do you need in order to design efficient algorithms to use this device? Do not mention detailed facts such as bit-level device register formats, but more basic qualities of the device such as capabilities, costs of certain operations, or operating constraints.

B2 Consider the programming techniques, algorithms, methods, tricks, etc. used in writing programs that interact with this type of device. What techniques, algorithms, methods, etc. have you used that require further knowledge of the device than is contained in the assumptions?

Questionnaire for Review C-1: Access Functions

Reviewer ________________

Interface ________________

Access Function ________________

C1 (Assumptions Allowing Implementation):

C2 (Applicability Conditions)

C3 (Effects Described?) . . . Yes ___ No ___

Discrepancies:

Questionnaire for Review C-2: Events Signalled

Reviewer ________________

Interface ________________

Event Signalled ____________________

C4 (Assumptions allowing Implementation):

C5 (Limitations Exist?) . . . . Yes ___ No ___

Limiting assumptions:

Questionnaire for Review C-3: Undesired Events

Reviewer ________________

Interface ________________

Undesired Event ______________________

C6 (Assumptions Allowing Detection):

C7 (Described in terms of virtual devices?) . . . . Yes ___ No ___

If not, what hidden aspects revealed?

Questionnaire for Review C-4: Local Data Types

Reviewer ________________

Interface ________________

Data Type ________________

C8 (Range):

C9 (Assumption defining possible values):

Questionnaire for Review C5: System Generation Parameters

Reviewer ________________

Interface ________________

System Generation Parameter __________________

C10 (Range):

C11 (Assumption defining possible values):

Questionnaire for Review C-6: Local Dictionary

Reviewer ________________

Interface _______________

Local Term __________________

C12 (Definition clear?) . . . . Yes ___ No ___

C13 (Confusion with other terms): . . . . . . Yes ___ No ___

C14 (Defined in terms of virtual device?): . . Yes ___ No ___

Questionnaire for Review C7: Completeness and Consistency

Reviewer ________________

Interface _______________

C15 (Assumptions not used):

C16 (Assumptions containing information not used):

C17 (Contradictory assumptions):

C18 (Unclear design decisions):

Questionnaire for Review D: Access Function Adequacy

Reviewer ______________ Date __________

Interface _____________

D1 (Can't Do's):

D2 (Additional Memory Needed):

D3 (Additional CPU Time Needed):

D4 (Unneeded Actions):

D5 (Prohibited Tricks):

D6 (Additional Functions Needed):

D7 (Functions Better Split):

D8 (Combinable Functions):

D9 (Virtual Device Preferable?):

D10 (Unreported Errors):

D11 (Reported But Unneeded Errors):

Questionnaire for Review E: Data Item/Function Index

Reviewer ____________________ Date __________

Requirements subsection____________________ Data Item ____________________

E1 (Sufficient Information?).......Yes ____ No ____

Additional Information Needed:

E2 (Unwanted Side-effects):

E3 (Unnecessary Actions):

REFERENCES

1 Heninger, K., Kallander, J., Parnas, D., and Shore, J.; Software Requirements for the A-7E Aircraft; NRL Memorandum Report 3876; 27 November 1978.

2 Crews, L., and Hall, C.; A-7D/E Aircraft Navigation Equations; NWC Technical Note 404-176; March 1975.

3 George, R.; A-7E Weapon Delivery Equations; NWC Technical Memorandum 2926; September 1976.

4 Reed, D., and Kanodia, R.; "Synchronization with Eventcounts and Sequencers"; Commun. of ACM, vol. 22, no. 2; February 1979.

5 Heninger, K., and Parnas, D.; Interface Specifications for the A-7 Extended Computer Module; in progress.

6 Parnas, D.; Use of Abstract Interfaces in the Development of Software For Embedded Computer Systems; NRL Report 8047; 1977.

7 D. Parnas and H. Wurges; "Response to Undesired Events in Software Systems"; Proc. Second Int. Conf. Software Eng., pp. 437-446; 1976.

8 Hoare, C. A. R.; "Monitors: An Operating System Structuring Concept"; Commun. of ACM, vol. 17, no. 10; October 1974.

COMMON ASSUMPTIONS FOR ALL DEVICES

Listed below are assumptions about undesired events that hold for all device interface modules in this document. Each assumption is followed by the Undesired Event that will be signalled if it is violated.

1. When user programs call access functions, all parameters will conform to the types specified in the access function tables in this document.

   (%Wrong type%)

2. For every matching value/effect function pair, user programs will not call the value access function ("G_") between the time the system is loaded and the first call to the associated effect access function ("S_").

   (%Undefined value%)

DI.1. AIR DATA COMPUTER (ADC)

DI.1.1. Introduction

The Air Data Computer (ADC) is a sensor that measures barometric altitude, true airspeed, and airspeed in relation to the speed of sound (Mach index). The device can be directed to perform a built-in test and to report the results.

DI.1.2. Interface Overview

DI.1.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G_ADC_ALTITUDE+	p1:distance;O	!+alt ADC+!	None
	p2:logical;O	!+ADC alt valid+!	
+G_ADC_MACH_INDEX+	p1:mach;O	!+mach ADC+!	
	p2:logical;O	!+ADC mach valid+!	
+G_ADC_TRUE_AIRSPEED+	p1:speed;O	!+TAS ADC+!	
	p2:logical;O	!+ADC TAS valid+!	
+S_ADC_SLP+	p1:pressure;I	sea level pressure	
+TEST_ADC+	p1:logical;O	!+ADC test result+!	

DI.1.2.2 Events Signalled

@T(!+ADC alt valid+!) @F(!+ADC alt valid+!)
@T(!+ADC mach valid+!) @F(!+ADC mach valid+!)
@T(!+ADC TAS valid+!) @F(!+ADC TAS valid+!)

DI.1.3. Basic Assumptions

1. The ADC provides measurements of the barometric altitude, true airspeed, and the mach number representation of airspeed of the aircraft (mach index). Any known measurement errors are compensated for within the module.

2. User programs are notified if the ADC cannot provide any of its outputs. If the access functions for barometric altitude, true airspeed, and mach index are called while valid data is unavailable most recent valid measurements (stale values) are provided.

4. The ADC is capable of performing a built-in test upon command from the software. The result of this test is returned to the software.

5. The minimum, maximum, and resolution of all ADC measurements are given by system generation parameters and are fixed after system generation time.

DI.1.4. Interface Design Issues

1. We have decided to provide reliability indicators with each function call in addition to an event that is signalled if the data becomes unavailable. See the discussion of problem 10 in the introduction to this chapter.

2. Earlier versions returned one reliability indicator for all three measurements (i.e. for the entire ADC). We have decided to return validity indicators with each data item (TAS, mach, and altitude) because it was suggested to us that some ADCs may fail to return one or two of these items, while still returning some good data.

3. We have decided that the ADC interface should return stale values for the barometric altitude, true airspeed, and mach index if the ADC hardware is unreliable (down). This is done for three reasons; (1) many programs make use of stale ADC values when the ADC is down, (2) if the software called an access function just as the unreliable event was being signalled, the stale value would still be valid, because the altitude, airspeed, or mach index would not have changed significantly, and (3) there is nothing better to do while the ADC is down.

DI.1.5. Local Data Types: None

DI.1.6. Local Dictionary

!+alt ADC+!	This is the ADC measured altitude above sea level of the aircraft.
!+ADC alt valid+!	If true then !+ADC alt+! is current. If false then !+ADC alt+! is the last available measurement.
!+ADC mach valid+!	If true then !+ADC mach+! is current. If false then !+ADC mach+! is the last available measurement.
!+ADC TAS valid+!	If true then !+ADC TAS+! is current. If false then !+ADC TAS+! is the last available measurement.
!+ADC test result+!	True indicates that the ADC built-in test has completed and detected no malfunctions. False indicates that a hardware malfunction has been detected.
!+mach ADC+!	This is the ADC measured mach index (the ratio of the aircraft's airspeed to the speed of sound in the surrounding air).
!+TAS ADC+!	This is the ADC measured true airspeed of the aircraft.

DI.1.7. Undesired Event Dictionary: None

DI.1.8. Function Effects

+S_ADC_SLP+	This function provides the ADC module with the sea level pressure to be used as a reference for altitude calculations.
+TEST_ADC+	Causes the ADC to initiate its self-test.

DI.1.9. System Generation Parameters

¢ADC_alt_max¢	Maximum altitude measurable with this ADC.
¢ADC_alt_min¢	Minimum altitude measurable with this ADC.
¢ADC_alt_res¢	ADC altitude measurement resolution.
¢ADC_mach_max¢	Maximum mach index measurable with this ADC.
¢ADC_mach_min¢	Minimum mach index measurable with this ADC.
¢ADC_mach_res¢	ADC mach index measurement resolution.
¢ADC_TAS_max¢	Maximum true airspeed measurable with this ADC.
¢ADC_TAS_min¢	Minimum true airspeed measurable with this ADC.
¢ADC_TAS_res¢	ADC true airspeed measurement resolution.

DI.1.10. Information Hidden

1. The scale, offset, I/O channel numbers, and format of the input data items.
2. The way in which the built-in test operates and the method used to determine if the test was passed or failed.
3. The operations that must be applied to the raw measurements from the device in order to produce correct barometric altitude, true airspeed, and mach number (reference (2)).
4. The relationship between the ADC and the FLR and the fact that both devices share the same initiate test command.

DI.1.11. Calculations Performed Within the Module

The barometric altitude is corrected within this interface by a function of vertical velocity and Mach index (reference (2)). The true airspeed is also corrected depending on the L-probe switch. No corrections are made to the Mach index value. These corrections are done within the abstract interface because they are dependent upon this particular ADC, the corrections make use of hidden data, and a different ADC might perform these corrections. In addition, the corrections can be done much more efficiently if they are incorporated with the scale and offset calculations.

DI.1.12. Implementation Notes

The ADC shares the initiate test command with the Forward Looking Radar (FLR). This means that when one of these devices is being tested, the other is unavailable for normal use (it is, in fact, also being tested whether the software wants the test result or not). For this reason the ADC and FLR device interfaces must operate through a monitor. If one of these devices is being tested then any calls to the other device must be held until the test completes. The +TEST_ADC+ and +TEST_FLR+ functions should keep the devices in test mode for only the minimum time required for the test.

DI.1R. AIR DATA COMPUTER RECONFIGURATION

DI.1.1R. Introduction

The specific ADC unit in a particular aircraft has one of two kinds of probes: L or non-L. The choice of probe affects the corrections applied to measurements within the module. The OFP must be parameterized so that type of probe can be changed without program reassembly or reloading. This section describes the access functions that allow this parameter to be read and written by user programs. These functions give away one secret of the actual device, i.e. that it has a changeable probe. Only a small set of other programs need them; programmers of the rest of the system need not know that they exist.

This description is a restricted supplement to the main specification. It uses terms and ideas defined in the ADC device interface description (DI.1). Each section is a restricted addenda to the corresponding sections of the main specification.

DI.1.2R. Interface Overview

DI.1.2.1R. Access Function Table

Function name	Parm type	Parm info	Undesired events
+G/S_ADC_LPROBE+	p1:logical;O	!+L-probe+!	None

DI.1.3R. Basic Assumptions

1. The ADC device interface module bases some of its calculations on an assumption about which of two possible probes (L-probe or not L-probe) is being used on this particular aircraft. The ADC module must permit this assumption to change at any time without requiring program reassembly or reloading.

2. If the probe assumed by the ADC module does not match the probe actually mounted on the aircraft, the ADC outputs are not guaranteed to meet their accuracy requirements.

DI.1.4R. Design Issues

1. We considered including these functions in the interface for the ADC, but felt that they gave away too much information about the actual device. As it is, only the programs that allow these parameters to be entered from the panel or displayed on the panel need to have access to these functions.

2. The G_ADC_LPROBE+ function is provided because of the requirement that the panel be able to display this data item. It would have been possible for the interface only to provide the +S ADC LPROBE+ function and require some other program to store the current probe for the panel display. Since the state must be kept inside the interface, it seems reasonable to store it in only place, providing functions to retrieve it for panel displays.

DI.1.5R. Local Types: None

DI.1.6R. Local Dictionary

!+L-probe+!	True indicates the ADC module is currently operating as it should for a device with an L probe; False means it is operating for a device with a non-L probe

DI.1.7R. Undesired Events: None

DI.1.8R. Function Effects

+S_ADC_LPROBE+	Future calls of +G_ADC_LPROBE+ return the value of p1; Future values returned by other ADC functions will be correct only if the value of p1 indicates the type of probe actually mounted on this aircraft

DI.1.9R. System Generation Parameters: None additional

DI.1.10R. Information hidden: None additional

DI.1.11R. Calculations performed within the module: None additional

DI.1.12R. Implementation notes: None additional

DI.2. ANGLE OF ATTACK SENSOR

DI.2.1. Introduction

The Angle of Attack device is a sensor that measures the angle of attack, that is, the angle between the Ya axis and the projection of the aircraft velocity vector into the Ya-Za plane. For a precise definition of the coordinate system involved, see appendix 6.

DI.2.2. Interface Overview

DI.2.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G_ANGLE_OF_ATTACK+	p1:angle;O	!+AOA+!	None
	p2:logical;O	!+AOA valid+!	

DI.2.2.2 Events Signalled: @T(!+AOA valid+!) @F(!+AOA valid+!)

DI.2.3. Basic Assumptions

1. The angle of attack value returned by this module is the true angle of attack. Any corrections required are done within this module.

2. The minimum, maximum, and resolution of angle of attack measurements are given by system generation time parameters.

3. The angle of attack measurement may not be available at all times. When data is requested the user program is informed if new data are available or not. If new measurements are not available then the last valid measurement is returned.

DI.2.4. Interface Design Issues: None

DI.2.5. Local Types: None

DI.2.6. Local Dictionary

!+AOA+!	This value is the angle between the aircraft Ya axis and the projection into the Ya-Za plane of the aircraft velocity vector.
!+AOA valid+!	True indicates !+AOA+! is current. False indicates !+AOA+! is the last valid measurement (stale).

DI.2.7. Undesired Event Dictionary: None

DI.2.8. Function Effects: None

DI.2.9. System Generation Parameters

¢AOA_max¢	The maximum angle of attack measureable with this sensor.
¢AOA_min¢	The minimum angle of attack measureable with this sensor.
¢AOA_res¢	The resolution of the angle of attack measurement.

DI.2.10. Information Hidden

1. The value encoding of the data words from the devices.

2. Corrections that are applied by this module.

DI.2.11. Calculations Performed Within the Module

Calculations performed within this module include performing corrections to the sensor measurements to obtain true angle of attack.

DI.2.12. Implementation Notes

The angle of attack value from the device must be read at a fixed rate, filtered with a first-order filter and corrected for a known bias (reference (2)).

AUDIBLE SIGNAL (Bomb Tone)

DI.3.1. Introduction

The Audible Signal device generates a tone that can be heard within the aircraft cockpit. The device has three states: on steady, on intermittently (beep), or off.

DI.3.2. Interface Overview

DI.3.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G/S_AUDIBLE_SIGNAL+	p1:ind_cntrl;O	!+Aud Signal+!	None
+S_BEEP_RATE+	p1:integer;I	rate (beeps/sec)	

DI.3.3. Basic Assumptions

1. There is one signal, controlled by the software, that can be heard by the pilot. This signal can be turned on steady, off, or it can alternate between on and off at a variable frequency. An example of this type of signal is a tone or whistle.

DI.3.4. Interface Design Issues: None

DI.3.5. Local Types

Type name	Type definition
ind_cntrl	enumerated: \$On\$, \$Off\$, \$Intermittent\$

DI.3.6. Local Dictionary:

!+Aud Signal+!	The current state of the audible signal.

DI.3.7. Undesired Event Dictionary: None

DI.3.8. Function Effects:

```
+S_AUDIBLE_SIGNAL+   if p1=$On$   then audible signal turned on
                                       p2 ignored;
                     if p1=$Off$  then audible signal turned off
                                       p2 ignored;
                     if p1=$Intermittent$ then audible signal alternated
                                               between on and off at rate
                                               specified by +S_BEEP_RATE+
```

+S_BEEP_RATE+	Sets the rate in beeps/second that the tone beeps when +S_AUDIBLE_SIGNAL+ is called with p1 = $Intermittent$.

DI.3.9. System Generation Parameters:

¢Beep_rate_default¢	The default beep rate for the audible signal.

DI.3.10. Information Hidden:

1. The value encoding of the data word to the device.

2. The method used to cause the signal to beep on and off.

DI.3.11. Calculations Performed Within the Module: None

DI.3.12. Implementation Notes: None

DI.4. COMPUTER FAIL SIGNAL

DI.4.1. Introduction

The Computer Fail device is a display and control device; it generates a signal to the pilot and several avionics devices which they interpret to mean that the computer system is unreliable.

DI.4.2. Interface Overview

DI.4.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired events
+G/S_COMPUTER_FAIL_SIGNAL+	P1:logical;O	!+Comp fail+!	None

DI.4.3. Basic Assumptions

1. The computer provides a signal which may be connected to any device and is interpreted by these devices as an indication of computer failure. The specific actions taken by these devices are given in reference (1). This command also causes a visual indicator to be turned on. The signal and the indicator cannot be changed independently.

DI.4.4. Interface Design Issues

The actions taken by other devices in response to the computer fail signal depend on the device itself and not the software. For this reason these actions are not described here.

DI.4.5. Local Types: None

DI.4.6. Local Dictionary

!+Comp fail+!	True indicates that the computer fail signal is on. False indicates indicates that the computer fail signal is off.

DI.4.7. Undesired Event Dictionary: None

DI.4.8. Function Effects

```
+S_COMPUTER_FAIL_SIGNAL+  if p1=true then computer fail signal turned on
                                          "Computer Fail" indicator turned on
                                          !+Comp fail+! will equal true;
                                     else computer fail signal turned off
                                          "Computer Fail" indicator turned off
                                          !+Comp fail+! will equal false;
```

DI.4.9. System Generation Parameters: None

DI.4.10. Information Hidden

1. The value encoding of the data words to the devices.

DI.4.11. Calculations Performed Within the Module: None

DI.4.12. Implementation Notes: None

DI.5. DOPPLER RADAR SET (DRS)

DI.5.1. Introduction

The Doppler Radar Set (DRS) is a sensor that measures aircraft ground speed and drift angle during flight.

DI.5.2. Interface Overview

DI.5.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired events
+G_DRS_DRIFT_ANGLE+	p1:angle;O	!+drift angle DRS+!	%Wrong mode%
+G_DRS_GROUND_SPEED+	p1:speed;O	!+gnd speed DRS+!	
+G_DRS_MODE+	p1:drs_mode;O	!+DRS mode+!	
+G_DRS_RELIABILITY+	p1:logical;O	!+DRS reliable+!	
+G_DRS_TEST_RESULT+	p1:logical;O	!+DRS test result+!	
+START_DRS+			
+STOP_DRS+			

DI.5.2.2 Events Signalled

@T(!+DRS reliable+!)   @F(!+DRS reliable+!)
@T(!+DRS mode changed+!)

DI.5.2.3 Modes of the Module

Mode name	Functions legal in the mode
!*OFF*!	+G_DRS_MODE+, +START_DRS+
!*OPERATE*!	+G_DRS_DRIFT_ANGLE+, +G_DRS_GROUND_SPEED+, +G_DRS_MODE+, +G_DRS_RELIABILITY+, +STOP_DRS+
!*MEMORY*!	+G_DRS_DRIFT_ANGLE+, +G_DRS_GROUND_SPEED+, +G_DRS_MODE+, +G_DRS_RELIABILITY+, +STOP_DRS+
!*STNDBY*!	+G_DRS_MODE+, +STOP_DRS+
!*TEST*!	+G_DRS_TEST_RESULT+, +G_DRS_MODE+, +G_DRS_RELIABILITY+, +STOP_DRS+

DI.5.3.1. Basic Assumptions

1. The DRS can measure the ground speed and the drift angle of the aircraft while in flight if certain conditions not under control of the software are met. The software is informed if these outputs are available.

2. The device has the following modes of operation: !*OFF*!, !*OPERATE*!, !*MEMORY*!, !*STNDBY*!, and !*TEST*!. An event can be signalled when the mode changes.

3. The ground speed and drift angle outputs are available when the mode is !*OPERATE*! or !*MEMORY*!.

4. During modes !*OFF*! and !*STNDBY*! no information, except mode, is available from the device.

5. If the DRS mode is !*OPERATE*! then the ground speed and drift angle measurements are current. If the DRS mode is !*MEMORY*! then the ground speed and drift angle measurements are the last valid measurements taken (often called stale values) because current measurements are not available due to flight or equipment conditions.

6. During mode !*TEST*! the software can use an access function to check the result of a built-in test.

7. The device may detect that it is not operating reliably at any time. An event can be signalled if this occurs.

8. The DRS module can be enabled and disabled by the software. The DRS module must be enabled to use any of the access functions, except the function that enables the module. Computer time is saved if the DRS module is disabled.

9. Range and resolution of all DRS measurements are given by system generation parameters and will not change without program reassembly.

DI.5.3.2. Assumptions about Undesired Events

1. The user programs will not call an access function in violation of mode table DI.5.2.3.

DI.5.4. Interface Design Issues

According to reference (2) the doppler ground speed must be corrected by a function of the barometric altitude. The correction is made within the interface. The alternative, performing the correction outside, was rejected because the correction is device dependent.

Both the smoothed and unsmoothed doppler outputs are used by the OFP. The smoothed outputs are used for display on the panel. Unfiltered outputs are used in all other parts of the OFP. The designers of this interface had two alternate ways to handle this. 1) Provide both filtered and unfiltered outputs. 2) Provide only unfiltered outputs and require the filtering be done externally. Alternative 2 was chosen because the filtering in this case is not device dependent.

We have included the mode !*STNDBY*! here because it seems likely that another DRS would also have a standby mode; which allows the pilot to interrupt radar transmissions for a period of time without turning the device completely off. The current software makes no use of the fact that the DRS is in standby mode, but it is not difficult to think of some cases where software action would depend on knowing if the DRS is in standby mode.

DI.5.5. Local Data Types

Type name	Type definition
drs_mode	enumerated: Off, $Operate$, $Memory$, $Stndby$, $Test$

DI.5.6. Local Dictionary

!+drift angle DRS+!	This value is the angle from the projection of the A/C Ya axis (appendix 6) onto the horizontal plane to the projection of the A/C velocity vector onto the horizontal plane. The angle is positive when measured CW from the projection of the A/C Ya axis (looking down). Thus, drift is positive when ground track is to the right of the A/C heading. When the mode is !*OPERATE*! this is the current measurement. When the mode is !*MEMORY*! this is a stale value.
!+DRS mode+!	The current operating mode of the DRS module.
!+DRS mode changed+!	True while the DRS mode is changing.
!+DRS reliable+!	True indicates that the DRS has not detected any internal malfunctions. False indicates that the DRS has detected an internal malfunction.
!+DRS test result+!	True indicates that the DRS has passed its built-in test. False indicates that it has failed the test.
!+gnd speed DRS+!	This value is the magnitude of the projection of the A/C velocity vector onto the horizontal plane. When the mode is !*OPERATE*! this is the current measurement. When the mode is !*MEMORY*! this is a stale value.

DI.5.7. Undesired Event Dictionary

%Wrong mode%	The DRS is operating in the wrong mode for the access function called.

DI.5.8. Function Effects

+START_DRS+	Changes the mode from !*OFF*! to one of the other modes. The mode entered depends on conditions beyond the control of the software, such as the pilot selected mode, the aircraft's flight characteristics, and the condition of the DRS device.
+STOP_DRS+	Sets the mode to !*OFF*!.

DI.5.9. System Generation Parameters

¢DRS_Drift_ang_max¢	The maximum drift angle measurable with this DRS.
¢DRS_Drift_ang_res¢	The resolution of the DRS drift angle measurement.
¢DRS_Gnd_speed_max¢	The maximum ground speed measurable with this DRS.
¢DRS_Gnd_speed_res¢	The resolution of the DRS ground speed measurement.

DI.5.10. Information Hidden

1. The data representation within the I/O words (scale and offset) is a secret of the DRS module.
2. The method used to determine if the ground speed and drift angle data words are valid.
3. The arrival sequence of the data.
4. Operations that must be done on the raw measurements from the device to compute the true ground speed and drift angle.
5. The method used to determine if the DRS has passed its built-in test, and what parts of that test operation are implemented in the hardware.
6. The rate at which the device is sampled.

DI.5.11. Calculations Performed Within the Module

The ground speed is corrected by a function of the barometric altitude (reference (2), page 6-16) within the module.

DI.5.12. Implementation Notes: None

DI.6. FLIGHT INFORMATION DISPLAYS (ADI, DME, HSI)

DI.6.1. Introduction

The Attitude Director Indicator (ADI), the Horizontal Situation Indicator (HSI), and the Distance Measuring Equipment (DME) are displays in the aircraft cockpit. An ADI device displays an elevation and an azimuth displacement from a fixed reference point. An HSI device is similar to a clockface; it displays two points on a circle relative to a fixed reference point. A DME device displays a string of decimal digits; it can be used to display an unsigned decimal number.

DI.6.2. Interface Overview

DI.6.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G_ADI_ELEV_AVAIL+	p1:logical;O	!+ADI elev avail+!	None
+G/S_ADI_AZIMUTH_INDICATOR+	p1:angle;O/I	!+ADI az+!	
+G/S_DME_DISPLAY+	p1:integer;O/I	!+DME display+!	
+G/S_DME_FLAG+	p1:logical;O/I	!+DME flag+!	
+G/S_HSI_POINTER_1+	p1:angle;O/I	!+HSI 1+!	
+G/S_HSI_POINTER_2+	p1:angle;O/I	!+HSI 2+!	
+G/S_ADI_ELEV_INDICATOR+	p1:angle;O/I	!+ADI elev+!	%ADI Elev not available%
+REMOVE_ADI_ELEV_INDICATOR+			

DI.6.2.2. Events Signalled

@T(!+ADI elev avail+!) @F(!+ADI elev avail+!)

DI.6.3. Basic Assumptions

1. The HSI is a device capable of displaying positions on a circle relative to a fixed reference position on the circle. For example, the positions may be indicated by pointers or lights around the circle. A clockface is an example of this type of display.

2. The ADI is a device capable of displaying elevation and azimuth displacements relative to a fixed reference point. The maximum and minimum displacements that can be indicated are available to the software and are fixed after system generation time. An attempt to indicate a displacement greater than the maximum (or less than the minimum) will result in the display of the maximum (or minimum) value.

3. The ADI elevation indicator is not available at all times. User programs can detect the availability of the indicator.

4. When under software control the ADI elevation indicator can be removed from the pilot's view by the software.

5. The DME display is capable of displaying an unsigned decimal integer value. There may be more that one physical display at different locations within the A/C, but all are connected together and receive the same value. All of these may not display the same number of digits. If the capacity is exceeded the information is presented in such a way that the pilot should be able to infer the missing digits from his flight conditions. Usually it is the most significant digit information that is not displayed.

6. There is an indicator flag that is controlled by the software visible on at least one of the DME displays.

7. The current value on the various displays is available from the module.

DI.6.4. Interface Design Issues

There is no function to remove the ADI azimuth indicator from view because the display will not move out of view.

DI.6.5. Local Data Types: None

DI.6.6. Local Dictionary

!+ADI az+!	The value currently being displayed on the ADI azimuth display.
!+ADI elev+!	The value currently being displayed by the ADI elevation display.
!+ADI elev avail+!	True indicates that the ADI elevation indicator is available for software control. False indicates otherwise.
!+DME display+!	The value currently being displayed on the DME display.
!+DME flag+!	True if the DME flag is visible, False otherwise.
!+HSI 1+!	The value currently being displayed by the HSI number 1 display.
!+HSI 2+!	The value currently being displayed by the HSI number 2 display.

DI.6.7. Undesired Event Dictionary

%ADI Elev not available%	The ADI elevation indicator is not available for software control because it is being controlled directly by some other avionics system.

DI.6.8. Function Effects

+REMOVE_ADI_ELEV_INDICATOR+	Causes the ADI elevation indicator to be removed from the pilot's view. It will remain out of view until the next call of +S_ADI_ELEV_INDICATOR+.
+S_ADI_AZIMUTH_INDICATOR+	Causes the indication of azimuth angle pl on the ADI. If pl is outside the range of azimuth angles that can be displayed then the maximum is indicated with the sign of pl.

+S_ADI_ELEV_INDICATOR+	if NOT !+ADI elev avail+!
	then
	UE(%ADI Elev not available%)
	else
	Causes the indication of elevation angle pl on the ADI. If pl is outside the range of elevation angles that can be displayed then the maximum is indicated with the sign of pl.

+S_DME_DISPLAY+	Causes the absolute value of pl to be output to all DME displays. The action in the event that the capacity of the display is exceeded is described by assumption 5 above.
+S_DME_FLAG+	Called with pl = true causes the flag associated with the DME displays to be displayed. Called with pl = false causes the flag to be removed from view.
+S_HSI_POINTER_1+	Causes HSI pointer 1 to indicate angle pl measured CW from the reference point as seen by the pilot.
+S_HSI_POINTER_2+	Causes HSI pointer 2 to indicate angle pl measured CW from the reference point as seen by the pilot.

DI.6.9. System Generation Parameters

¢ADI_azimuth_max¢	The maximum azimuth displacement that can be shown with the ADI azimuth indicator.
¢ADI_azimuth_min¢	The minimum azimuth displacement that can be shown with the ADI azimuth indicator.
¢ADI_elevation_max¢	The maximum elevation displacement that can be shown with the ADI elevation indicator.

¢ADI_elevation_min¢	The minimum elevation displacement that can be shown with the ADI elevation indicator.
¢ADI_res¢	Resolution of the ADI indicators.
¢HSI_res¢	Resolution of the HSI indicators.

DI.6.10. Information Hidden

1. The value encoding of the data to the devices.
2. The maximum displayable values on the individual DME displays.
3. The actual limits of the ADI elevation and azimuth indicators hidden until system generation time.

DI.6.11. Calculations Performed Within the Module

The calculations performed are (1) dealing with scale and offset of the data items, (2) adjustments to the integer for display on the DME digits if it exceeds the size capacity of any displays, and (3) limiting the ADI indicators to their maximum displacements.

DI.6.12. Implementation Notes: None

DI.7. FORWARD LOOKING RADAR (FLR)

DI.7.1. Introduction

The Forward Looking Radar (FLR) is used to measure distances and to display a point on a radar screen. It operates very differently in its four mutually exclusive modes, which are called !*RANGING*!, !*CDCE*! (Computer Driven Cursor Enabled), !*TF*!, and !*IDLE*!. All of the modes except TF are entered under user program control; TF is entered under internal control and overrides any other modes. In RANGING mode, the FLR is a sensor that measures the slant range from the aircraft to a given ground location. In CDCE mode, the software controls the position of two cursors on the radar display screen. In TF mode, no user program actions are legal. In IDLE mode, the FLR can be directed to perform a built-in test and to report the results.

DI.7.2. Interface Overview

DI.7.2.1. Access Functions

Function name	Parm type	Parm info	Undesired Events
+G_FLR_CURSOR_LIMITS+	p1:distance;O	!+Rng cursor min+!	%FLR in TF%
	p2:distance;O	!+Rng cursor max+!	
	p3:angle;O	!+Az cursor lft max+!	
	p4:angle;O	!+Az cursor rgt max+!	
+G_FLR_DIRECTION_LIMIT+	p1:angle;O	!+Dir az min+!	
	p2:angle;O	!+Dir az max+!	
	p3:angle;O	!+Dir el min+!	
	p4:angle;O	!+Dir el max+!	
+S_FLR_MODE+	p1:flr_mode;I		
+G_FLR_LOCKED_ON+	p1:logical;O	!+FLR locked on+!	%Wrong FLR mode%
+REMOVE_FLR_ AZIMUTH_CURSOR+			
+G/S_FLR_AZIMUTH_CURSOR+	p1:angle;O/I	!+Az cursor+!	
+G/S_FLR_DIRECTION+	p1:angle;O/I	!+FLR az+!	
	p2:angle;O/I	!+FLR elev+!	
+G/S_FLR_RANGE_CURSOR+	p1:distance;O/I	!+Rng cursor+!	
+SLEW_FLR_CURSORS+	p1:real;I	Range slew	
	p2:real;I	Azimuth slew	
+TEST_FLR+	p1:logical;O	!+FLR test result+!	
+G_FLR_RANGE+	p1:distance;O	!+Slt range FLR+!	%Wrong FLR mode% %FLR not locked on%
+G_FLR_MODE+	p1:flr_mode;O	!+FLR mode+!	None

DI.7.2.2 Events Signalled

@T(!+FLR in terrain following+!) @F(!+FLR in terrain following+!)
@T(!+FLR locked on+!) @F(!+FLR locked on+!)

DI.7.2.3. Modes of the Module

Mode name	Functions legal in the mode
!*CDCE*!	+S_FLR_MODE+, +INHIBIT_AZIMUTH_CURSOR+, +S_CURSOR_POSITION+
!*IDLE*!	+S_FLR_MODE+, +TEST_FLR+
!*RANGING*!	+S_FLR_MODE+, +G_FLR_LOCKED_ON+, +G_FLR_RANGE+, +S_FLR_DIRECTION+
!*TF*!	None

After power-up the module is in mode !*IDLE*!. The module may be in only one mode at a time.

DI.7.3.1. Basic Assumptions

1. The FLR has four mutually exclusive modes of operation; these modes are named !*RANGING*!, !*CDCE*!, !*TF*! (Terrain Following), and !*IDLE*!.

2. In mode !*RANGING*! the FLR is a sensor that can measure the distance from the aircraft to a given target on the ground (this distance is called the slant range). The target is identified by elevation and azimuth angles given in the airframe coordinate system (see appendix 6). The FLR must be locked-on to the target before a slant range measurement may be made. Software can detect if the FLR is locked-on to the target.

3. In mode !*CDCE*! the software can control the position of two cursors on the radar display screen. One cursor is positioned in azimuth (left or right of the screen center) and the other is positioned on the screen to show a specified ground range from the aircraft. The software can either specify cursur positions directly, or use a slew function to move the cursors on the screen.

4. There is a mode of FLR operation, called Terrain Following (!*TF*!). This mode is entered without any action by the OFP regardless of the current FLR mode. Other programs are notified when !*TF*! is entered and exited. The software cannot change the mode from !*TF*! to any other mode. When !*TF*! ends the mode will become !*IDLE*!.

5. It is possible for the FLR to be in none of the above modes, in this case the mode is !*IDLE*!. While in this mode it is possible for the software to initiate a built-in test of the FLR. The result of the test is reported to the software.

6. The minimum, maximum, and resolution of the slant range are given as system generation time parameters and will not change without reassembly of the program.

7. The limits of the FLR cursors and the pointing angle are not necessarily fixed at execution time. The software can access the current limits.

8. The FLR module will require some commonly available information concerning the current aircraft situation (velocity and attitude information). The module will call access programs in other modules to get this information. The exact items required depend upon the implementation.

9. The current position of the display cursors and the FLR pointing direction are available from the module.

DI.7.3.2. Assumptions about Undesired Events

1. User programs will not call access functions in violation of mode table DI.7.2.3.

DI.7.4. Interface Design Issues

1. The FLR antenna is roll-stabilized (i.e. it is rotated about its Yr axis as the A/C rolls in order to keep its Xr axis parallel to the IMS platform X axis). Most often the FLR antenna is positioned by angles calculated in the airframe coordinate system and a future FLR might use this coordinate system. For this reason the abstract interface assumes angles in the airframe coordinate system and correction to the actual system is done internally.

2. During !*TF*! the current FLR requires drift angle, flight path angle, and ground speed information. This information is used by the hardware, but is not displayed directly. We have chosen not to provide access functions to set these items, but to assume that the module calls access programs somewhere else (like the physical model or the data banker) to get them. This was done so that the items required could be a secret of the module; in much the same way that the data required for sensor corrections is in some other modules. If the FLR were changed, the particular items of information required by it could also change.

3. The Master Function Switch (MFS) button labeled "TF" is considered part of the FLR and interpreted by this module. Earlier versions of this interface and the Swtich Bank Interface showed the MFS as it is in the requirements; one device. We have decided to split the master function switch between the FLR and the Weapon Release System module, because the MFS "TF" position deals primarily with the FLR and the other MFS positions deal with weapon release.

4. This interface has been designed to support run-time changes in the values of the cursor ranges and the FLR direction limits. If the FLR hardware does not have changeable limits then the functions will simply return the fixed limits. The "worst case" (in terms of required memory space) value is available at assembly time.

DI.7.5. Local Data Types

Type name	Type definition
flr_mode	enumerated: $Ranging$, $Idle$, $Cursor$, $TF$

DI.7.6. Local Dictionary

!+Az cursor+!	The azimuth currently being displayed by the FLR cursors.
!+Az cursor lft max+!	The maximum displacement of the FLR azimuth cursor to the left side of the display screen as seen by the pilot.
!+Az cursor rgt max+!	The maximum displacement of the FLR azimuth cursor to the right side of the display screen as seen by the pilot.
!+Dir az max+!	The maximum displacement of the FLR pointing angle along the Xr axis in the positive direction.
!+Dir az min+!	The maximum displacement of the FLR pointing angle along the Xr axis in the negative direction.
!+DIR el max+!	The maximum displacement of the FLR pointing angle along the Zp axis in the positive direction.
!+Dir el min+!	The maximum displacement of the FLR pointing angle along the Zp axis in the negative direction.
!+FLR az+!	The azimuth angle of the FLR pointing direction.
!+FLR elev+!	The elevation angle of the FLR pointing direction.
!+FLR in terrain following+!	True if mode is !*TF*! False otherwise
!+FLR locked on+!	True indicates that the FLR is locked on to a target and range information is available. False indicates otherwise.
!+FLR mode+!	Current mode of the FLR.
!+FLR test result+!	True indicates that the FLR has passed its built-in test.
!+Rng cursor+!	The range currently being displayed by the FLR cursors.

!+Rng cursor max+!	The maximum ground range from the aircraft displayable with the FLR range cursor.
!+Rng cursor min+!	The minimum ground range from the aircraft displayable with the FLR range cursor.
!+Slt range FLR+!	The slant range from the aircraft to the locked on target as measured by the FLR.

DI.7.7. Undesired Event Dictionary

%FLR in TF%	The mode cannot be changed while the mode is !*TF*!. The program must await the signal that !*TF*! has ended.
%FLR not locked on%	revealed by +G_FLR_LOCKED_ON+: p1 = true shows inhibited; p1 = false shows enabled
%Wrong FLR mode%	Occurs if an access function is called when in an improper mode for that function. Refer to table section 2.3.

DI.7.8. Function Effects

+REMOVE_FLR_AZIMUTH_CURSOR+	Removes the azimuth cursor from the screen until the next call to +S_FLR_CURSOR_POSITION+.
+S_FLR_AZIMUTH_CURSOR+	The azimuth cursor is positioned on the radar display screen by angle p1. This angle is measured from the centerline of the display to the azimuth cursor. Positive angles are measured clockwise looking at the display.
+S_FLR_DIRECTION+	Causes the FLR to be pointed at a target for ranging by angles p1 and p2. Parameter p1 specifies the azimuth angle and p2 specifies the elevation angle. Let the FLR pointing vector be any vector that points away from the A/C in the FLR pointing direction. The FLR pointing angles are given in the airframe coordinate system. The azimuth angle is the angle between the A/C Ya axis and the projection of the FLR pointing vector into the Ya-Xa plane. This angle is positive measured CW looking down onto the A/C. The elevation angle is the angle between the FLR pointing vector and its projection into the Ya-Xa plane. This angle is positive when the Za component of the FLR pointing vector is positive, i.e. pointing up is positive when the A/C is level. If an out of range value is given the FLR is pointed to the extreme of its limits in the correct direction.

+S_FLR_RANGE_CURSOR+	The range cursor is positioned depending on the value of p2. The cursor will appear on the radar screen superimposed at the locus of points that is p2 distance from the aircraft.
+S_FLR_MODE+	This function changes the mode of the module. It cannot be called while the mode is !*TF*!.
+SLEW_FLR_CURSORS+	The FLR range cursor is moved at the rate ¢FLR_rng_slew_rate¢ * pl and the azimuth cursor is moved at the rate ¢FLR_az_slew_rate¢ * p2. The range cursor moves outward when pl is positive. The azimuth cursor moves to the right (as seen by the pilot) when p2 is positive. A value of one for pl or p2 causes the cursors to move at th maximum rate.
+TEST_FLR+	The built-in test of the FLR is initiated. Parameter pl returns the result of the test. See !+FLR test result+! for interpretation of value.

DI.7.9. System Generation Parameters

¢FLR_az_cursor_res¢	Resolution of the azimuth cursor position parameter.
¢FLR_az_slew_rate¢	The maximum slew rate of the azimuth cursor.
¢FLR_dir_az_res¢	Resolution of the azimuth pointing angle.
¢FLR_dir_elev_res¢	Resolution of the elevation pointing angle.
¢FLR_rng_slew_rate¢	The maximum slew rate of the range cursor.
FLR_slt_range_max¢	Maximum slant range measurable by this FLR.
¢FLR_slt_range_min¢	Minimum slant range measurable by this FLR.
¢FLR_slt_range_res¢	Resolution of the FLR slant range measurement.
¢FLR_rng_cursor_res¢	Resolution of the range cursor position parameter.

DI.7.10. Information Hidden

1. The scale, offset, and format of the FLR I/O data words.

2. The way in which the FLR is pointed at a target, including the fact that there is a mechanically steered antenna, rather than an electronically directed beam. Facts about the FLR antenna hidden include the particular sequence of operations necessary to position the FLR antenna, the coordinate system transformation that must be done, and the antenna slave command.

3. The way that the software can detect that the pilot has set the FLR mode to Terrain Following.

4. The particular items of information required by the FLR during !*TF*! and how they are used.

5. The relationship between the FLR and other devices, such as the ADI and the ADC.

7. The details of the FLR built-in test and the method used to determine if the hardware passes the test or not.

DI.7.11. Calculations Performed Within the Module

The FLR pointing angles must be transformed from aircraft body coordinates to the FLR antenna coordinate system.

DI.7.12. Implementation Notes

1. The Master Function switch is connected directly to the FLR and when /MFSW/ = TF (reference (1)) the FLR hardware goes into terrain following mode.

2. The FLR shares the initiate test command with the Air Data Computer (ADC). This means that when one of these devices is being tested, the other is unavailable for normal use (it is, in fact, also being tested whether the software wants the test result or not). For this reason the ADC and FLR device interfaces must operate through a monitor. If one of the devices is being tested, then any calls to the other device must be held until the test is complete. +TEST_ADC+ and +TEST_FLR+ should keep the devices in the test mode for the minimum time required for the test.

DI.8. HEAD-UP DISPLAY (HUD)

DI.8.1. Introduction

The Head-Up Display (HUD) is a display device that projects information into the pilot's field of vision so that he can see it without looking down in the cockpit. The HUD displays two kinds of symbols: location indicators, which can be positioned under software control, and value indicators, which are always in the same place and display a value provided by software.

DI.8.2. Interface Overview

DI.8.2.1 Access Function Table

In the table s1 = AS, ASL, FLTDIR, FPM, PUAC, USC, WARN
s2 = AS, FPM, LSC, PUAC, USC

Function name	Parm type	Parm info	Undesired events
+G_HUD_RELIABILITY+	p1:logical;O	!+HUD reliable+!	%In test%
+G/S_HUD_s1_MODE+	p1:ind_cntrl;O/I	!+s1 mode+!	
+G/S_HUD_s2_POSITION+	p1:angle;O/I	!+s2 elevation+!	
	p2:angle;O/I	!+s2 azimuth+!	
+G/S_HUD_ASL_POSITION+	p1:angle;O/I	!+ASL elevation+!	
	p2:angle;O/I	!+ASL azimuth+!	
	p3:angle;O/I	!+ASL rotation+!	
+G/S_HUD_FLTDIR_POSITION+	p1:angle;O/I	!+FLTDIR azimuth+!	
+G/S_HUD_NACC_DISPLAY+	p1:real;O/I	!+HUD NACC+!	
+G/S_HUD_ALT_DISPLAY+	p1:real;O/I	!+HUD alt+!	
+G/S_HUD_HEADING_DISPLAY+	p1:angle;O/I	!+HUD heading+!	
+G/S_HUD_PITCH_DISPLAY+	p1:angle;O/I	!+HUD pitch+!	
+G/S_HUD_ROLL_DISPLAY+	p1:angle;O/I	!+HUD roll+!	
+G/S_HUD_VERTVEL_DISPLAY+	p1:real;O/I	!+HUD vertvel+!	
+SLEW_HUD_AS+	p1:real;I	AS horiz rate	
	p2:real;I	AS vert rate	
+G/S_HUD_LSC_MODE+	p1:ind_cntrl;O/I	!+LSC mode+!	%RNGCUE on% %In test%
+G/S_HUD_PUC_MODE+	p1:ind_cntrl;O/I	!+PUC mode+!	%On illegal% %In test%
+G/S_HUD_RNGCUE_MODE+	p1:ind_cntrl;O/I	!+RNGCUE mode+!	%LSC on% %In test%
+G/S_HUD_VVFPM_MODE+	p1:ind_cntrl;O/I	!+VVFPM mode+!	%Blink illegal% %In test%
+G/S_HUD_TEST_MODE+	p1:HUD_test;O/I	!+HUD test mode+!	None

DI.8.2.2 Events Signalled

@T(!+HUD reliable+!) @F(!+HUD reliable+!)

DI.8.3. Basic Assumptions

1. The HUD is a device capable of displaying information and symbols in the pilot's field of view as he looks forward out of the aircraft. Some of the symbols can be positioned by the software, while others are always at a fixed position. The software can turn some symbols on, off, or cause them to blink.

2. The software has two-dimensional control (azimuth and elevation) over the following symbols:

   Aiming Symbol (AS)
   Azimuth Steering Line Center Point (ASL)
   Flight Path Marker (FPM)
   Lower Solution Cue (LSC)
   Upper Solution Cue (USC)
   Pull-Up Anticipation Cue (PUAC)

   The software has one-dimensional control (azimuth) over the flight director (FLTDIR). The positioning of these symbols is relative to the optical reference axes (ORA) on the display screen. Azimuth angles are measured from the vertical ORA and are positive to the right when looking at the HUD from the pilot's viewpoint. Elevation angles are measured from the horizontal ORA and are positive upward when looking at the HUD from the pilot's viewpoint. The vertical ORA is parallel to the Za axis. The horizontal ORA is parallel to the Xa axis.

   The In Range Cue (RNGCUE) is always displayed superimposed on the Aiming Symbol.

   In addition to control over its center point the software can rotate the ASL through 360 degrees. It is rotated about its center point. The rotation angle is zero when the ASL is parallel to the vertical ORA and is measured CW when looking at the HUD from the pilot's viewpoint.

3. The warning symbol (WARN) and the pullup-cue (PUC) are displayed at fixed positions when turned on.

4. All symbols except the PUC and FPM can be off, blinking, or on steady. The PUC cannot be held on steady. The LSC and the RNGCUE cannot be displayed simultaneously.

5. In addition to symbols the HUD displays quantitative information including vertical velocity, normal acceleration, altitude, heading, pitch, and roll.

6. The altitude, heading, vertical velocity, and normal acceleration information might not always be displayed, but the software cannot detect if they are being displayed.

7. The HUD has a test mode which causes the display of one of two test patterns on the display screen. When the test is initiated nothing is displayed on the screen except the selected test pattern. When the test is exited the HUD is returned to the exact state that it was in before the test was started. That is all symbols and indicators are reset to their previous state and position.

8. The range of movement of all movable symbols and the range of the quantitative displays are given by system generation parameters.

9. The position of all movable symbols and the values being displayed on the quantitative displays are available from the module.

DI.8.3.2. Assumptions about Undesired Events

1. When the HUD self test function is active the user programs will not call any access functions except +G_HUD_TEST_MODE+ and +S_HUD_TEST_MODE+.

2. User programs will not enable the LSC and the RNGCUE at the same time.

3. User programs will not attempt to cause the vertical velocity indicator and flight path marker (VVFPM) to blink.

4. User programs will not attempt to cause the PUC to be displayed on steady.

DI.8.4. Design issues for the Interface

1. In-range cue. During several weapon delivery modes the lower solution cue is used as an "in-range" cue (reference (a)). When so used the lower solution cue is superimposed on the aiming symbol. A function is provided that implements the in-range cue; i.e. turn on the lower solution cue and position it over the aiming symbol. The cue is maintained over the aiming symbol wherever the latter is positioned. The simultaneous use of the lower solution cue and the in-range cue is not permitted. There is a practical advantage to this approach. The programs that use the in-range cue simply need to turn it on when the specified conditions have been met. Without this function they would need to explicitly position the lower solution cue over the aiming symbol. This involves more work for the program and the location of the aiming symbol may not be known to the program that determines the in-range conditions.

2. Aiming symbol slewing. Functions are provided that support slewing of the aiming symbol. Slewing refers to the ability to cause the symbol to move across the display at a specified rate. It is not neccessary to continuously specify the position of the symbol, only its rate of movement along each of the two axes. The abstract interface programs keep track of the symbol position. The requirements do not require this type of slewing for any other HUD symbol and this capability could easily be added for any other symbol if required in the future.

3. Earlier versions of this interface provided only set functions for the positions and modes of the symbols (except the aiming symbol). We have decided to provide both set and get functions for these items to allow programs that did not set symbol states to detect the state of the symbols.

DI.8.5. Local Data Types

Type name	Type definition
ind cntrl	enumerated: \$On\$, \$Intermittent\$, \$Off\$
HUD test	enumerated: \$A\$, \$B\$, \$None\$

DI.8.6. Local Dictionary

!+HUD alt+!	The most recent value given to the module to display on the HUD altitude display.
!+HUD heading+!	The most recent value given to the module to display on the HUD heading display.
!+HUD NACC+!	The most recent value given to the module to display on the HUD normal acceleration dislay.
!+HUD roll+!	The most recent value given to the module to display on the HUD roll display.
!+HUD reliable+!	True if HUD is ready and has passed its built-in test. False if HUD is either not ready or has failed its built-in test. This state does not ensure that symbols are not being displayed; only that the HUD may have failed.
!+HUD test mode+!	True if the HUD is in the test mode, false otherwise.
!+HUD vertvel!	The most recent value given to the module to display on the HUD vertical velocity display.
!+AS azimuth+! !+FPM azimuth+! !+LSC azimuth+! !+PUAC azimuth+! !+USC azimuth+! !+ASL azimuth+! !+FLTDIR azimuth+!	Present azimuth angle of the symbol relative to the HUD ORA as seen by the pilot.

!+AS elevation+! !+FPM elevation+! !+LSC elevation+! !+PUAC elevation+! !+USC elevation+! !+ASL elevation+!	Present elevation angle of the symbol relative to the HUD ORA as seen by the pilot.
!+AS mode+! !+ASL mode+! !+FLTDIR mode+! !+FPM mode+! !+LSC mode+! !+PUAC mode+! !+PUC mode+! !+RNGCUE mode+! !+USC mode+! !+VVFPM mode+! !+WARN mode+!	Present display mode of the symbol.
!+ASL rotation+!	Rotation angle of the ASL. The angle is zero if the ASL is parallel to the vertical ORA. The angle is 90 degress if the ASL is parallel to the horizontal ORA.

DI.8.7. <u>Undesired Event Dictionary</u>

%Blink illegal%	This symbol may not be blinked. It must be ofi or held on steady.
%In test%	The HUD is currently conducting a software requested self-test and no other functions are available.
%LSC on%	Enabled by +S_HUD_LSC_MODE+ with pl=On or $Blink$ Inhibited by +S_HUD_LSC_MODE+ with pl=Off
%On illegal%	This symbol may not be held on steady. It must blink or be off.
%RNGCUE on%	Enabled by +S_HUD_RNGCUE_MODE+ with pl=On or $Blink$ Inhibited by +S_HUD_RNGCUE_MODE+ with pl=Off

DI.8.8. Function Effects

+S_HUD_AS_MODE+	Called with pl=On causes the appropriate symbol to be displayed steady. pl=$Intermittent$ causes the symbol toblink. pl=Off removes the symbol from the display.
+S_HUD_ASL_MODE+	
+S_HUD_FPM_MODE+	
+S_HUD_LSC_MODE+	
+S_HUD_USC_MODE+	
+S_HUD_WARN_MODE+	
+S_HUD_PUAC_MODE+	
+S_HUD_FLTDIR_MODE+	
+S_HUD_RNGCUE_MODE+	
+S_HUD_TEST_MODE+	Called with pl=A or B causes the display of the selected test pattern. All other active symbols are removed. Called with pl=$None$ causes the pattern to be removed and all symbols are restored to their state before the test.
+S_HUD_PUC_MODE+	Called with pl=$Intermittent$ causes the PUC to be displayed blinking. Called with pl=Off causes the symbol to be removed from the display. The PUC cannot be displayed steady.
+S_HUD_AS_POSITION+	These functions are used to position these symbols on the display. The symbol is elevated, displaced in azimuth, and rotated by the parameters. Note that all parameters are not used for all of these functions. These functions may be called while the symbol is off. When turned on, the symbol will appear at the last position specified. If any of the angles are out of range then the symbol is displayed at the nearest edge to where it would be shown if the screen were large enough.
+S_HUD_ASL_POSITION+	
+S_HUD_FPM_POSITION+	
+S_HUD_LSC_POSITION+	
+S_HUD_USC_POSITION+	
+S_HUD_PUAC_POSITION+	
+S_HUD_FLTDIR_POSITION+	
+S_HUD_ASL_ROTATION+	
+S_HUD_ALT_DISPLAY+	These functions provide the altitude, heading, pitch, roll, vertical velocity, and normal acceleration information to the HUD for display.
+S_HUD_HEADING_DISPLAY+	
+S_HUD_PITCH_DISPLAY+	
+S_HUD_ROLL_DISPLAY+	
+S_HUD_VERTVEL_DISPLAY+	
+S_HUD_NACC_DISPLAY+	
+S_HUD_VVFPM_MODE+	Called with pl=true causes the vertical velocity/acceleration display to function and the FPM to be under software control. Called with pl=false inhibits function of these displays and prevents software control of the FPM.

+SLEW_HUD_AS+	The aiming symbol is moved horizontally at the rate ¢HUD_slew_rate¢ * pl, and vertically at the rate ¢HUD_slew_rate¢ * p2. The AS moves upward and to the right, as seen by the pilot, for positive values of pl and p2. A value of one for pl and p2 cause the AS to move at its maximum rate. If an attempt is made to move the aiming symbol out of visual range, then the symbol is displayed at the edge nearest to where it would be shown if the screen were large enough.

DI.8.9. System Generation Parameters

¢HUD_alt_max¢	The maximum value displayable on the HUD altitude display.
¢HUD_alt_min¢	The minimum value displayable on the HUD altitude display.
¢HUD_alt_res¢	The resolution of the HUD altitude display.
¢HUD_angular_res¢	The resolution of all HUD angular parameters (e.g. symbol azimuth/elevation, HUD roll, pitch, etc.).
¢HUD_blink_rate¢	The rate that the HUD symbols blink on and off when in that mode.
¢HUD_fltdir_az_max¢	The maximum azimuth displacement of the HUD flight director symbol.
¢HUD_nacc_max¢	The absolute value of the maximum/minimum normal acceleration displayable on the HUD.
¢HUD_nacc_res¢	The resolution of the HUD normal acceleration display parameter.
¢HUD_slew_rate¢	The maximum slew rate that the AS can be slewed across the display screen (degrees/second).
¢HUD_symbol_az_max¢	The maximum azimuth displacement from the ORA of the following symbols: AS, ASL, FPM, LSC, USC, and PUAC.
¢HUD_symbol_el_max¢	The maximum elevation displacement from the ORA of the following symbols: AS, ASL, FPM, LSC, USC, and PUAC.
¢HUD_vertvel_max¢	The absolute value of the maximum/minimum vertical velocity displayable on the HUD.
¢HUD_vertvel_res¢	The resolution of the HUD vertical velocity display parameter.

DI.8.10. Information Hidden

1. The particular sequence of operations neccessary to enable and position the various HUD symbols.

2. The scale and offset of the data words for numeric displays.

3. The method used to generate the symbols, i.e. which symbols are hardware produced and which are produced by software actions such as superimposing basic symbols.

4. The reason for restrictions on certain symbols, such as the RNGCUE and LSC being exclusive, and the PUC must flash when on.

5. Which symbols have hardware controlled blinking, and the method used to causes certain symbols to be removed from the display.

6. How the HUD test pattern is generated.

7. How to continuously position the aiming symbol so it looks like it is moving smoothly at a given rate.

DI.8.11. Calculations Performed Within the Module

The only calculations performed within the HUD abstract interface are simple ones involving symbol positions.

DI.8.12. Implementation Notes

1. Some of the symbols have no hardware function to turn them on or off, or cause them to blink. However all symbols can be removed from the screen by positioning them beyond their maximum. Symbols without direct on/off controls will be turned off by moving them from view. Symbols can be made to blink by moving them in and out of view rapidly (or turning them on and off if such control is available).

DI.9. INERTIAL MEASUREMENT SET (IMS)

DI.9.1. Introduction

The IMS is a sensor that can be used to measure aircraft attitude and velocity components. The three orthogonal axes of the IMS define a coordinate system that is described in appendix 3.

DI.9.2. Interface Overview

DI.9.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired events
+G_IMS_MAG_HEADING+	p1:angle;O	!+heading MAG+!	none
+G_IMS_MAG_VARIATION+	p1:angle;O	!+magvar IMS+!	
+G_IMS_MODE+	p1:imsmode;O	!+IMS mode+!	
+G_IMS_STATUS+	p1:logical;O	!+IMS ready+!	
	p2:logical;O	!+IMS rel+!	
+S_IMS_ENABLE+	p1:logical;I	!+IMS enable+!	
+S_IMS_SCALE+	p1:imsscale;I	!+IMS scale+!	
+G_IMS_PITCH+	p1:angle;O	!+pitch IMS+!	%Coarse rotation in
+G_IMS_ROLL+	p1:angle;O	!+roll IMS+!	progress%
+G_IMS_TRUE_HEADING+	p1:angle;O	!+heading IMS+!	
+G_COARSE_ROTATING+	p1:logical;O	!+IMS rotating+!	%IMS disabled%
+S_X_COARSE_ROTATION+	p1:angle;I		
+S_Y_COARSE_ROTATION+	p1:angle;I		
+S_Z_COARSE_ROTATION+	p1:angle;I		
+G_IMS_E_VELOCITY+	p1:speed;O	!+E vel IMS+!	%Vel not init%
+G_IMS_N_VELOCITY+	p1:speed;O	!+N vel IMS+!	%IMS disabled%
+G_IMS_V_VELOCITY+	p1:speed;O	!+V Vel IMS+!	%Coarse rotation in progress%
+S_IMS_E_VELOCITY+	p1:speed;I		%Coarse rotation in
+S_IMS_N_VELOCITY+	p1:speed;I		progress%
+S_IMS_V_VELOCITY+	p1:speed;I		%IMS disabled%
+S_X_FINE_ROTATION+	p1:angle;I		
+S_Y_FINE_ROTATION+	p1:angle;I		
+S_Z_FINE_ROTATION+	p1:angle;I		

DI.9.2.2 Events Signalled

@T(!+IMS mode changed+!)	
@T(!+IMS rel+!)	@F(!+IMS rel+!)
@T(!+IMS ready+!)	@F(!+IMS ready+!)

DI.9.3.1. Basic Assumptions

1. There is an inertial platform with three orthogonal axes and a coordinate system that is defined by the platform.

2. IMS devices of a given model vary slightly from each other. These variations depend on the actual device, not on the abstract device described in this interface. An implementation for the IMS abstract interface must be parameterized so that it can be reconfigured to serve any device of a given model without program reassembly or reloading. Although the IMS module must provide access functions to allow these parameters to be set, the parameters are not visible in the abstract interface description because they show the nature of the actual device. The initialization functions are described in the IMS reconfiguration section. Users of the IMS abstract interface must assume that they have been set; if not, the abstract device will not operate correctly.

3. User programs can control one aspect of the IMS operating status:
   - whether the IMS is enabled or disabled for computer control.

   Three other aspects of the operating status depend on the activities within the interface or on inputs from the hardware:
   - the ready and reliable status of the hardware,
   - the operating mode of the device,
   - whether coarse rotations are in progress.

   User programs can discover the last three aspects from the IMS abstract interface.

4. There are other devices in the aircraft avionics system connected directly to the IMS. These devices assume that the platform is aligned to Earth north, east, and vertical. Some of the other avionic systems may require that the platform be aligned this way in order to function properly.

5. Since the three axes of the platform are rigidly attached to each other, an error in the alignment of one axis may affect the measurements taken on the other two.

6. When the IMS is enabled it is possible to rotate the inertial platform around any of its three axes in one of two ways; fine movements with closer accuracy or coarse movements with a larger range.

7. It is assumed that fine rotations are used to keep the platform aligned in some particular way, while coarse rotations are used to change the alignment of the platform. For this reason fine rotations appear to occur immediately after being requested. That is they affect the outputs sensitive to platform orientation as if the rotation were performed immediately.

8. During coarse rotations certain IMS functions are not available. It is possible for user programs to find out if any coarse rotations are being done. Request for coarse rotations are accumulated, i.e., if a rotation of N degrees is requested and then a rotation of M degrees is requested about the same axis the total rotation will be N+M degrees about that axis. Rotations are stopped and accumulated requests are lost if the IMS is disabled, turned off, becomes unreliable, or becomes unready.

9. The abstract device can measure the incremental aircraft velocity component along any of the three axes. Therefore, given an initial velocity the device can determine the current velocity component along each of the three axes. The velocity components must be reinitialized after the device is disabled and then reenabled, turned off and then back on, becomes reliable after being unreliable, or ready after being unready. These measurements are sensitive to platform alignment (see assumption 7).

10. North and east velocity components can be measured on one of two scales: a fine scale with a small resolution and a coarse scale with a larger resolution. Users of the IMS module can choose which scale they want to use.

11. The abstract device can measure the angle between an aircraft axis and the corresponding platform axis. These measurements are sensitive to platform alignment (see assumption 7).

12. The abstract device also returns the magnetic heading from the magnetic compass and the magnetic variation entered by the pilot into the IMS control panel. The alignment of the platform does not affect these values. New values of the magnetic variation are available only when the IMS is unreliable. These items are not updated if the IMS is turned off.

13. The angles and velocity components returned by the module will be stale values if the IMS internal status prevents them from being updated (e.g. IMS off or unreliable). User programs can check status programs to determine whether the values are fresh. However there may be a time lag between when the values stop being updated and when the status programs register the changes. This lag will not be long enough for the IMS values to change significantly; stale values can be used safely during this period.

14. The abstract device can measure true heading and magnetic heading from 0 to 360 degrees. Pitch measurements range from ± 90 degrees and roll measurements range from ± 180 degrees. Fine platform rotations are limited to ± 2 degrees, and coarse platform rotations can be done within the full 360 degree range. The range of velocity measurements are given by system generation parameters, as are the resolutions of all values.

DI.9.3.2. Assumptions about Undesired Events

1. User programs will not attempt to access pitch, roll, true heading, or velocity information while a coarse platform rotation is in progress.

2. User programs will not attempt to set intitial velocity information or initiate fine platform roations while a coarse platform rotation is in progress.

3. User programs will not attempt to initiate a fine or coarse platform rotation if the IMS is disabled.

4. User programs will not attempt to set initial velocity values or access current velocity if the IMS is disabled.

5. User programs will not attempt to access velocity values without first setting initial values since the IMS was last enabled (with +S_IMS ENABLE+).

DI.9.4. Interface Design Issues

1. We have rejected the alternative of having the interface indicate a "wander-azimuth" IMS. Such a design would have had the effect of hiding a great many of the functions in the requirements document (ref. 1) within the abstract interface module. The basic idea of the abstract interface module is to hide rather minor differences between possible devices. Hiding major differences would mean a very complex module. Wander azimuth would be such a major difference. In addition wander azimuth is impossible because certain other devices in the A-7 are directly connected to the IMS and assume that it is aligned. Therefore maintaining the platform in actual alignment is a requirement of the OFP. We will, in general, stay close to the terms used in the A-7 requirements document in designing the abstract interfaces.

2. The fact that the outputs respond immediately to fine rotations mean that the time required to "torque" the actual platform will be hidden inside the module. The readings will be corrected to adjust for rotations that have not yet taken place. This can be done relatively simply and results in the fact that no other program needs to know the rotation rate of the platform. This differs from wander azimuth because the corrections are actually applied to the platform. The outputs are only corrected by the amount of error left in the platform due to the time required for the physical rotation of the platform. Outputs to other devices directly from the IMS (i.e. not through the computer) will not change instantaneously.

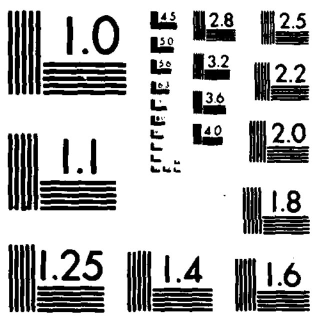
3. We have not hidden the coarse (slewing) rotations in the same way. Providing accurate outputs during slewing is neither easy nor useful. Most functions cannot be performed while slewing is going on and the abstract interface does not attempt to hide this.

AD-A092 696 NAVAL RESEARCH LAB WASHINGTON DC F/G 9/2
ABSTRACT INTERFACE SPECIFICATIONS FOR THE A-7E DEVICE INTERFACE--ETC(U)
NOV 80 R A PARKER, K L HENINGER, D L PARNAS
UNCLASSIFIED NRL-MR-4385 NL

2 of 2

AD A
092696

END
DATE
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

4. We decided to have the abstract interface return IMS velocities rather than incremental velocities because the incremental velocities are never used outside of this module and calculation of the platform velocities is simple. We do not produce other outputs such as position because these computations are more complex and involve more than one sensor value.

5. We earlier considered an interface that interpreted calls to certain functions when the IMS was not ready, unreliable, or off as undesired events. UEs are reserved for conditions the program can control or be absolutely sure about. This is necessary because of our requirement that the program not depend on UEs to function correctly in the production version. Under the older version, even if the user programs preceded each IMS access function with a call to +G_IMS_STATUS+ and +G_IMS_MODE+ to be sure that the IMS was reliable, ready, and on the UE could occur because the conditions are beyond the control of the software so that they may possibly change between the time the program checked them and the call to the access function.

   The solution proposed here is to have the access functions (roll, pitch, true heading, magnetic heading, and velocities) return stale values if the IMS internal status prevents them from being updated. These conditions will be monitored frequently by the IMS module and if they change an event will be signalled (see section 2.2). The user programs will use +G_IMS_STATUS+ or +G_IMS_MODE+ to determine why the event was signalled and then take the appropriate action. If it is assumed that the event is signalled not long after one of the conditions changes then the stale value will cause no harm. The maximum time permitted between the change in condition and the event would be related to the rate that the IMS values can change significantly.

6. We have decided to hide the fact that this IMS needs to know if the latitude is greater than 70 degrees. The module is designed assuming that it can get such information from other access programs. This was done since the need to know when the latitude is greater than 70^{o} is a characteristic of this particular IMS and that information has no effect on the software use of the IMS.

DI.9.5. Local types

Type name	Type definition
imsmode	enumerated: \$Gndal\$, \$Norm\$, \$Iner\$, \$Magsl\$, \$Grid\$, \$Off\$
imsscale	enumerated: \$Fine\$, \$Coarse\$

DI.9.6. Local dictionary

!+E vel IMS+!	The component of the aircraft velocity along the Xp or East axis as measured by the IMS. The value is positive in the positive Xp direction.
!+heading IMS+!	The angle measured clockwise in the Xp-Yp plane (looking in the negative Z direction - down) from the Yp axis to the projection in the Xp-Yp plane of the Ya axis.
!+heading MAG+!	The magnetic heading of the aircraft. Magnetic heading is the angle measured CW from magnetic north (looking down) to the horizontal component of the Ya axis. This value is not affected by the alignment of the platform.
!+IMS enable+!	The value last set by +S_IMS_ENABLE+
!+IMS mode+!	The current setting of the IMS mode switch.
!+IMS mode changed+!	True while the IMS mode is being changed. False otherwise.
!+IMS ready+!	True iff IMS is ready for operation under computer control.
!+IMS rel+!	True iff the IMS is to be considered reliable based on its internal self-test.
!+IMS rotating+!	True iff coarse rotation in progress.
!+IMS scale+!	The value last set by +S_IMS_SCALE+
!+magvar IMS+!	The value input by the pilot into the IMS control panel.
!+N vel IMS+!	The component of the aircraft velocity along the Yp or North axis as measured by the IMS. The value is positive in the positive Yp direction.
!+pitch IMS+!	The angle between the Ya axis and its projection into the Xp-Yp plane. The angle is negative when the positive Ya axis is below the Xp-Yp plane, and positive otherwise.
!+roll IMS+!	The angle between the Xa axis and its projection into the Xp-Yp plane. The angle is positive when the positive Xa axis is below the Xp-Yp plane (negative Zp component) and negative otherwise.
!+V vel IMS+!	The component of the aircraft velocity along the Zp or Vertical axis as measured by the IMS. The value is positive in the positive Zp direction.

DI.9.7. Undesired Event Dictionary

%IMS disabled%	The IMS is not enabled for computer control. enabled: @T(power up) affected by +S_IMS_ENABLE+: enabled when p1 = false; inhibited when p1 = true
%Vel not init%	The IMS velocity requested has not been given an initial velocity. enabled: @T(power up) enabled by +S_IMS_ENABLE+ with p1 = false enabled: @F(!+IMS rel+!) enabled: @F(!+IMS ready+!) enabled: @T(!+IMS mode changed+!) if !+IMS mode+! = Off inhibited by associated +S_IMS_?_SPEED+ when %IMS disabled% is inhibited
%Coarse rotation in progress%	The platform is currently under coarse rotation. During this time velocity and attitude access functions are disabled. enabled by any of +S_?_COARSE_ROTATION+ with p1 not equal to zero; inhibited when rotations complete; revealed by +G_COARSE_ROTATING+; p1 = false shows inhibited; p1 = true shows enabled

DI.9.8. Function Effects

+S_IMS_E_VELOCITY+ +S_IMS_N_VELOCITY+ +S_IMS_V_VELOCITY+	These functions initialize the IMS velocities and inhibit %Vel not init%. Future values returned by +G_IMS_?_VELOCITY+ will be based on p1 and the velocity changes that have been measured since the latest +S_IMS_?_VELOCITY+ was called.
+S_IMS_ENABLE+	This function enables the IMS device for computer control. When called with p1=true %IMS disabled% is inhibited. When called with p1=false %IMS disabled% and %Vel not init% are enabled, %Coarse rotation in progress% is inhibited, coarse rotations are stopped, and accumulated coarse rotation requests are lost.
+S_IMS_SCALE+	When called with p1=$Fine$ the finer scale is used for north and east velocity measurements. When called with p1=$Coarse$ the coarser scale is used for north and east velocity measurements. The vertical velocity measurements are not affected by this function.

+S_X_FINE_ROTATION+ +S_Y_FINE_ROTATION+ +S_Z_FINE_ROTATION+ +S_X_COARSE_ROTATION+ +S_Y_COARSE_ROTATION+ +S_Z_COARSE_ROTATION+	The inertial platform is rotated about the specified axis by angle p1. A positive angle causes the platform to rotate CW looking along the axis from the origin. Calling +S_?_COARSE_ROTATION+ will enable %Coarse rotation in progress%.

DI.9.9. System Generation Parameters

¢IMS_attitude_res¢	The resolution of the IMS pitch, roll, and heading measurements.
¢IMS_coarse_rot_res¢	The resolution of IMS platform coarse rotation angles.
¢IMS_coarse_vel_res¢	The resolution of coarse IMS velocity measurements.
¢IMS_E_vel_max¢	The maximum velocity magnitude measurable in the east direction.
¢IMS_fine_rot_res¢	The resolution of fine platform rotation angles.
¢IMS_fine_vel_res¢	The resolution of fine IMS velocity measurements.
¢IMS_N_vel_max¢	The maximum velocity magnitude measurable in the north direction.
¢IMS_V_vel_max¢	The maximum velocity magnitude measurable in the positive vertical direction.

DI.9.10. Information Hidden

1. The particular sequence of operations neccessary to provide torquing/slewing commands to the platform gyros. The users of the abstract interface request a correction in terms of an angle of rotation about a particular axis. The abstract interface produces the correct gyro torquing/slewing commands.
2. The scale and offset of input and output data items (reference (1)).
3. The format of the input data words.
4. The particular corrections that are applied in computing the interface output values.
5. The algorithm that produces current platform velocities from the IMS incremental velocities and an initial velocity.
6. The fact that the IMS mode switch input is the same for "off" and "none".

7. The amount of time taken to perform fine rotations of the platform.

8. How it is determined that the IMS is ready for computer control.

9. The fact that this IMS is sensitive to the latitude of the aircraft

DI.9.11. Calculations Performed Within the Module

The IMS abstract interface calculates and maintains the east, north, and vertical platform velocities. The velocity returned by the access functions includes compensation for the known accelerometer bias and coriolis acceleration. The IMS abstract interface calls an access function in the physical model module for the coriolis acceleration at the current A/C position.

Corrections are applied to the platform angles, the known bias in the incremental velocity readings is compensated for, and the adjustment to the platform required to compensate for drift is computed.

DI.9.12. Implementation Notes

Fine rotations must interact correctly with coarse rotations in order to make the interface behave as if fine rotations occur instantaneously. The user cannot know if fine rotations are in progress when he requests a coarse rotation. Therefore coarse rotations are adjusted to compensate for fine rotations that the caller requested earlier, but are still in progress.

DI.9R. INERTIAL MEASUREMENT SET RECONFIGURATION

DI.9.1R. Introduction

The specific IMS unit in a particular aircraft has individual characteristics that must be taken into account by the IMS device interface module. These characteristics include measurement errors and calibration parameters. The IMS device interface module is parameterized so that the values of these parameters can be changed without program reassembly or reloading. This section describes the access functions that allow these parameters to be read and written by other parts of the software. Since these functions give away the nature of the actual device behind the interface, they are not included in the IMS device interface description (DI.9). Only a small set of other programs need them; programmers of the rest of the system need not know that they exist.

This description supplements the previous section. It uses terms and ideas defined in the IMS device interface description (DI.9).

DI.9.2R. Interface Overview

DI.9.2.1R. Access Function Table

All the entries in this table represent value/effect function pairs.

Function name	Parm type	Parm info	UEs
+G/S_E_COARSE_VEL_BIAS+	p1:real;O/I	!+E coarse bias+!	None
+G/S_E_COARSE_VEL_SCALE+	p1:real;O/I	!+E coarse scale+!	
+G/S_E_FINE_VEL_BIAS+	p1:real;O/I	!+E fine bias+!	
+G/S_E_FINE_VEL_SCALE+	p1:real;O/I	!+E fine scale+!	
+G/S_N_COARSE_VEL_BIAS+	p1:real;O/I	!+N coarse bias+!	
+G/S_N_COARSE_VEL_SCALE+	p1:real;O/I	!+N coarse scale+!	
+G/S_N_FINE_VEL_BIAS+	p1:real;O/I	!+N fine bias+!	
+G/S_N_FINE_VEL_SCALE+	p1:real;O/I	!+N fine scale+!	
+G/S_V_VEL_BIAS+	p1:real;O/I	!+V fine bias+!	
+G/S_V_VEL_SCALE+	p1:real;O/I	!+V fine scale+!	
+G/S_X_GYRO_DRIFTRATE+	p1:real;O/I	!+X drift+!	
+G/S_X_GYRO_SCALE+	p1:real;O/I	!+X corr increm+!	
+G/S_Y_GYRO_DRIFTRATE+	p1:real;O/I	!+Y drift+!	
+G/S_Y_GYRO_SCALE+	p1:real;O/I	!+Y corr increm+!	
+G/S_Z_GYRO_DRIFTRATE+	p1:real;O/I	!+Z drift+!	
+G/S_Z_GYRO_SCALE+	p1:real;O/I	!+Z corr increm+!	

DI.9.3R. Basic Assumptions

1. The parameters characterizing an individual IMS are numerical. Their range and resolution are given as system generation parameters.

2. Each IMS axis drifts at a rate that can be considered constant. The rate varies from unit to unit, and may change somewhat during the lifetime of a particular unit. This drift can be compensated for by the virtual device module so that the axes of the virtual device appear to have no drift.

3. During fine rotations, the platform is rotated around a particular axis in small fixed increments. The increments may vary from axis to axis on a single platform. The size of the increment is constant for an individual IMS unit, but vary between different IMS units.

4. A raw IMS measurement must be multiplied by a scale factor to obtain an incremental velocity. For a given platform, this scale factor varies from axis to axis; for a given axis, it varies depending on whether the IMS is operating with fine or coarse scale (see DI.9, S_IMS_SCALE). The scale factor values remain constant for an individual IMS unit, but vary between different IMS units.

5. A raw IMS velocity measurement includes a predictable measurement error (bias) that can be compensated for by the IMS device interface module. On a given platform, the size of this bias varies from axis to axis; it varies for a given axis depending on whether the IMS is operating in fine or coarse scale. The bias values remain constant for an individual IMS unit, but vary between different IMS units.

6. An IMS may be replaced (or its parameters changed) without program reassembly being required.

DI.9.4R. Design Issues

1. The "+G_+" functions are provided because of the requirement that the panel be able to display these data items. It would have been possible for the interface only to provide the "+S_+" functions and require some other program to store the current values for the panel display. Since the values must be kept inside the interface, it seems reasonable to store them in only one place, providing functions to retrieve them for panel displays.

DI.9.5R. Local Types: None

DI.9.6R. Local Dictionary

!+E coarse bias+!	Measurement error for the East axis when the velocities are being measured by the coarse scale.
!+E coarse scale+!	Scale factor used for velocity calculation for the East axis when the velocities are being measured by the coarse scale.
!+E fine bias+!	Measurement error for the East axis when the velocities are being measured by the fine scale.
!+E fine scale+!	Scale factor used for velocity calculation for the East axis when the velocities are being measured by the coarse scale.
!+N coarse bias+!	Measurement error for the North axis when the velocities are being measured by the coarse scale.
!+N coarse scale+!	Scale factor used for velocity calculation for the North axis when the velocities are being measured by the coarse scale.
!+N fine bias+!	Measurement error for the North axis when the velocities are being measured by the fine scale.
!+N fine scale+!	Scale factor used for velocity calculation for the North axis when the velocities are being measured by the coarse scale.
!+V fine bias+!	Measurement error for the vertical axis when the velocities are being measured by the fine scale.
!+V fine scale+!	Scale factor used for velocity calculation for the vertical axis when the velocities are being measured by the coarse scale.
!+X corr increm+!	Size of rotation increments for the X axis during fine correction
!+X drift+!	Drift rate for the X axis.
!+Y corr increm+!	Size of rotation increments for the Y axis during fine correction
!+Y drift+!	Drift rate for the Y axis.
!+Z corr increm+!	Size of rotation increments for the Z axis during fine correction
!+Z drift+!	Drift rate for the Z axis.

DI.9.7R. <u>Undesired Event Dictionary</u>: none

DI.9.8R. <u>Function Effects</u>

<u>Directly visible effects</u>:

The functions in this interface come in "G_"/"S_" pairs. After a call to "S_" function with p1 = n, calls to corresponding "G_" function return the value n.

<u>Additional effects</u>

The "S_" functions below affect the correct functioning of the "G_" functions listed beside them, that is, the "G_" function is not guaranteed to return an accurate value unless the "S_" function has been called with the correct value for that particular IMS device on that aircraft.

+S_E_COARSE_VEL_BIAS+	+G_IMS_E_VELOCITY+ when IMS scale is coarse
+S_E_COARSE_VEL_SCALE+	
+S_E_FINE_VEL_BIAS+	+G_IMS_E_VELOCITY+, when the IMS scale is fine
+S_E_FINE_VEL_SCALE+	
+S_N_COARSE_VEL_BIAS+	+G_IMS_N_VELOCITY+, when the IMS scale is coarse
+S_N_COARSE_VEL_SCALE+	
+S_N_FINE_VEL_BIAS+	+G_IMS_N_VELOCITY+, when the IMS scale is fine
+S_N_FINE_VEL_SCALE+	
+S_V_VEL_BIAS+	+G_IMS_V_VELOCITY+
+S_V_VEL_SCALE+	

The "S_" functions below affect the alignment of the platform. Thus, unless these functions have been called with the correct values for the particular IMS in the aircraft, correct platform cannot be achieved. This means none of the "G_" functions can be guaranteed to provide accurate values, and other devices in the aircraft will not operate correctly (see DI.9, assumption 4).

+S_X_GYRO_DRIFTRATE+
+S_Y_GYRO_DRIFTRATE+
+S_Z_GYRO_DRIFTRATE+

+S_X_GYRO_SCALE+
+S_Y_GYRO_SCALE+
+S_Z_GYRO_SCALE+

DI.9.9R. System Generation Parameters

¢IMSR_coarse_bias_max¢	Maximum, minimum, and resolution respectively of
¢IMSR_coarse_bias_min¢	the coarse bias parameters
¢IMSR_coarse_bias_res¢	
¢IMSR_coarse_vscale_max	Maximum, minimum, and resolution respectively of
¢IMSR_coarse_vscale_min	the coarse scale parameters
¢IMSR_coarse_vscale_res	
¢IMSR_corr_inc_max¢	Maximum, minimum, and resolution respectively of
¢IMSR_corr_inc_min¢	the corr. increm. parameters
¢IMSR_corr_inc_res¢	
¢IMSR_drift_max¢	Maximum, minimum, and resolution respectively of
¢IMSR_drift_min¢	the drift parameters
¢IMSR_drift_res¢	
¢IMSR_fine_bias_max¢	Maximum, minimum, and resolution respectively of
¢IMSR_fine_bias_min¢	the coarse bias parameters
¢IMSR_fine_bias_res¢	
¢IMSR_fine_vscale_max¢	Maximum, minimum, and resolution respectively of
¢IMSR_fine_vscale_min¢	the fine scale parameters
¢IMSR_fine_vscale_res¢	

DI.9.10R. Information hidden: see DI.9

DI.9.11R. Calculations performed within the module: see DI.9

DI.9.12R. Implementation notes: see DI.9

Panel Interface

DI.10.1. Introduction

The panel is a data entry and display device. The data entry component consists of a keyboard that can be used to enter digits and selected letters. The display component has a few special purpose indicators and three windows that can be used to display alphanumeric strings.

DI.10.2. Interface Overview

DI.10.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired events
+CLEAR_LOWER+			
+CLEAR_MARK+			
+CLEAR_UPPER+			
+G/S_E_LIGHT+	p1:logical;O/I	!+E light+!	
+G/S_ENTER_LIGHT+	p1:logical;O/I	!+Enter light+!	
+G_KEYBD_INPUT+	p1:keybd;O	!+Keybd input+!	
+G/S_KEYBD_ENTER_LIGHT+	p1:logical;O/I	!+Keybd enter light+!	
+G/S_LOWER_DEC+	p1:logical;O/I	!+Decimal pt+!	
+G/S_LOWER_FORMAT322+	p1:logical;O/I	!+Format L322+!	
+G/S_N_LIGHT+	p1:logical;O/I	!+N light+!	
+G/S_S_LIGHT+	p1:logical;O/I	!+S light+!	
+G/S_UPPER_FORMAT222+	p1:logical;O/I	!+Format U222+!	
+G/S_UPPER_FORMAT321+	p1:logical;O/I	!+Format U321+!	
+G/S_W_LIGHT+	p1:logical;O/I	!+W light+!	
+S_LOWER_WINDOW+	p1:char_string(6);I		
+S_MARK_WINDOW+	p1:char_string(1);I		
+S_UPPER_WINDOW+	p1:char_string(7);I		

DI.10.2.2. Events Signalled

@T(!+Enter pressed+!)	@F(!+Enter pressed+!)
@T(!+Keybd pressed+!)	@F(!+Keybd pressed+!)
@T(!+Keybd input ready+!)	
@T(!+Mark pressed+!)	@F(!+Mark pressed+!)

DI.10.3. Basic Assumptions

1. The panel is a device capable of receiving input from an operator and transmitting this information to the computer program. The panel also can display information from the program to the operator via several alphanumeric displays (called windows) and indicators associated with the windows.

2. The panel has three separate displays called the upper, lower, and mark windows. These displays have six, seven, and one character positions respectively.

3. There are two indicators on the upper window labeled "N" and "S" and two indicators on the lower window labeled "E" and "W". These indicators can be turned on and off by the calling programs.

4. The panel interface will signal when any of three buttons, labeled "Enter", "Keybd", and "Mark" are pressed or released.

5. There is an indicator associated with the Enter and Keybd buttons. There is one command to turn on the Enter indicator and one command to turn on both indicators. It is not possible to turn the Keybd indicator on alone.

6. It is possible to display data in the upper windows in one of the following formats:

XX'XX'XX"	called format U222
XXX'XX'X	called format U321
XXXXXX	no format indicators on

It is possible to display data in the lower window in one of the following formats:

XXX'XX'XX"	called format L322
XXX'XX'.XX"	called format L322 with decimal
XXXXX.XX	decimal point on
XXXXXXX	no format indicators on

The mark window has no format indicators associated with it.

7. In addition to the buttons already named the panel contains a set of data buttons. An event is signalled when any of the data buttons are pressed. An access function will return the identity of the pressed button.

8. If the operator presses a data button before the program gets the previous keyboard entry from the interface the data is lost and an error condition is indicated.

DI.10.4. <u>Interface Design Issues</u>

1. This interface does not attempt to abstract away from the three display windows and the size of the windows. This panel is quite close to the A-7 requirements (reference 1). It is likely that any major change to the panel would be accompanied by major changes in the requirements so nothing would be gained by providing a very abstract panel. On the other hand there would be a significant increase in complexity and perhaps loss of run-time efficiency. This interface effectively isolates the software from minor changes to the panel, such as value encodings. Changes of this nature would not normally be associated with changes in the requirements. This interface also simplifies the use of the panel by the various parts of the OFP.

2. The error condition that can occur if the operator presses another data button before the software has sampled the previous keystroke is not treated here as an undesired event because the condition can occur in the production system without indicating a programming error. It is not difficult to produce this error in the current version of the system (NWC-2). If we treated it as a UE we would be in effect guaranteeing that some program could respond to the event @T(!+Keybd input ready+!) in the least possible time between keystrokes.

3. An earlier attempt to hide the rules for panel entry and the display format for different types of data resulted in a very complex interface. We decided to design the interface to hide less and be closer to the actual device. The more complex functions that hide the entry rules and display formats will be in another module.

DI.10.5. Local Data Types

Type	Type definition
keybd	enumerated: $None$, 0, 1, $N2$, $L3$, $W4$, $H5$, $-E6$, $C7$, $S8$, $D9$, $Error$
char_string(n)	character string of n length.

DI.10.7. Local Dictionary

!+Enter pressed+!	True if the push button labeled "Enter" is pressed.
!+Format L322+!	True if display format 322 for the lower window is on.
!+Format U222+!	True if display format 222 for the upper window is on.
!+Format U321+!	True if display format 321 for the upper window is on.
!+Keybd input+!	Identifies the data button pressed since the last time +G_KEYBD_INPUT+ was called. Equals $Error$ if more than one data button has been pressed since the last time +G_KEYBD_INPUT+ was called.
!+Keybd input ready+!	True when !+Keybd input+! not equal $None$.
!+Keybd pressed+!	True if the push button labeled "Keybd" is pressed.
!+Mark pressed+!	True if the push button labeled "Mark" is pressed.

For all of the following, true means that the associated indicator is turned on, false means that it is turned off.

Term	Associated Indicator
!+Decimal pt+!	the decimal point on the lower window
!+E light+!	"E" indicator
!+Enter light+!	indicator associated with the Enter button
!+Keybd-Enter light+!	both the indicator associated with the "Keybd" button and the indicator associated with the "Enter" button
!+N light+!	"N" indicator
!+S light+!	"S" indicator
!+W light+!	"W" indicator

DI.10.7. Undesired Event Dictionary: None.

DI.10.8. Function Effects

+CLEAR_LOWER+ +CLEAR_MARK+ +CLEAR_UPPER+	These blank out all of the data in the specified window.
+S_E_LIGHT+ +S_N_LIGHT+ +S_S_LIGHT+ +S_W_LIGHT+	These functions control the display of the indicators "N" and "S" on the upper window, and the "W" and "E" on the lower window. A call with pl=true turns on the display, pl=false turns off the display. !+? light+! is set to pl; where ? is E, N, S, or W.
+S_ENTER_LIGHT+	Called with pl=true the indicator associated with the "Enter" button is turned on; pl=false causes the indicator to be turned off if !+Keybd-Enter light+! = false, else it remains on. !+Enter light+! is set to pl.
+S_KEYBD_ENTER_LIGHT+	Called with pl=true causes both the "Enter" and "Keybd" indicator to go on; pl=false causes the "Keybd" indicator to go off. If !+Enter light+! = false then the "Enter" indicator also will go off, else it will remain on. !+Keybd-Enter light+! is set to pl.
+S_LOWER_WINDOW+ +S_MARK_WINDOW+ +S_UPPER_WINDOW+	The character string given by pl is displayed on the specified window. If the length of the character string is less than the size of the window the data is displayed right-justified with blank fill. If the length of the character string is greater than the size of the window then the data is displayed right-justified and the extra characters are truncated.
+S_LOWER_DEC+	The decimal point on the lower window is turned on if pl=true; else it is turned off. !+Decimal pt+! is set to pl.
+S_UPPER_FORMAT222+	Called with pl=true causes the display of data in the upper window to be of the format XX'XX"XX'. Any previous display format for the upper window is ended. !+Format U222+! is set to true, !+Format U321+! is set to false.
+S_UPPER_FORMAT321+	Called with pl=true causes the display of data in the upper window to be of the format XXX'XX"X'. Any previous display format for the upper window is ended. !+Format U321+! is set to true, !+Format U222+! is set to false,
+S_LOWER_FORMAT322+	Called with pl=true causes the display of data in the lower window to be of the format XXX'XX"XX'. !+Format L322+! is set to true.

DI.10.9. System Generation Parameters: None.

DI.10.10. Information Hidden

1. The value encoding of the data from/to the panel hardware, including the operations required to generate the characters on the displays.

DI.10.11. Calculations Performed Within the Module: None.

DI.10.12. Implementation Notes: None.

DI.11. PROJECTED MAP DISPLAY SET (PMDS)

DI.11.1.1. Introduction

The Projected Map Display Set (PMDS) is a display device that shows a map of an area surrounding a specified geographic location. The specified location can be positioned under one of two reference points on the map display screen. Maps of several different scales may be displayed. Scale, reference point, and position selection are under software control.

DI.11.2. Interface Overview

DI.11.2.1.1. Access Function Table (Device Control)

Function name	Parm type	Parm info	Undesired Events
+DISPLAY_MAP_WARNING+			None
+G_MAP_DISPLAYABLE+	p1:latitude;I	map latitude	
	p2:longitude;I	map longitude	
	p3:logical;O	!+Map displayable+!	
+G/S_MAP_INDICATOR+	p1:angle;O/I	!+Map indicator+!	
+G/S_MAP_POINTER_ANGLE+	p1:angle;I	!+Map pointer ang+!	
+G_MAP_POSITION+	p1:logical;O	!+Map position valid+!	
	p2:latitude;O	!+Map latitude+!	
	p3:longitude;O	!+Map longitude+!	
+G/S_MAP_REFERENCE_PT+	p1:ref_pt;I	!+Map ref pt+!	
+G/S_MAP_ROTATION+	p1:angle;I	!+Map rotation+!	
+G/S_MAP_SCALE+	p1:map_scale;I	!+Map scale+!	
+SLEW_MAP+	p1:real;I	slew up down	
	p2:real;I	slew right left	
+S_MAP_POSITION+			%Not displayable%

DI.11.2.1.2. Access Function Table (Map scale Data Type Operations)

Function name	Parm type	Parm info	Undesired Events
+NEXT_SCALE+	p1:map_scale;I/O		%Already grossest%
+PREV_SCALE+	p1:map_scale;I/O		%Already finest%
+SCALES_EQUAL+	p1:map_scale;I		None
	p2:map_scale;I		
	p3:logical;O	!+scales equal+!	
+SCALES_GREATER+	p1:map_scale;I		
	p2:map_scale;I		
	p3:logical;O	!+scales greater+!	
+SCALES_LESSTHAN+	p1:map_scale;I		
	p2:map_scale;I		
	p3:logical;O	!+scales lessthan+!	

DI.11.3.1. Basic Assumptions

1. The PMDS is a device capable of displaying maps of various scales with a specified geographic location under a reference point on the map display screen. The requested scale, location, and reference point adequately specifies the display wanted.

2. The reference point can be set by the user of this module. The reference point is the point on the map display screen that the requested map location is positioned under. The reference point can be set to any legal value of type "ref_pt".

3. The number of map scales and the finest and grossest scale values are known and will not change after program assembly. The map scale values have a total ordering.

4. The user of this module can set the current scale of the map to any legal value of type "map_scale".

5. The available maps might not include all geographic locations. The user of this module must determine that a given location can be displayed at a given scale before attempting to position the map.

6. There may be locations that can be displayed in more than one way for a particular scale. When there is such a choice the module chooses the way yielding the least delay.

7. The map can be rotated on the display screen in order to orient the map on the display screen in any way desired. The map rotation is the angle between the line from the center of the display to north on the map, and the line from the center of the display screen to the top-center of the display. When used with the map display screen the directions top, bottom, right, and left refer to those directions as seen by the pilot when the plane is upright.

8. There is a pointer displayed on the screen. The pointer can be rotated about the center of the display screen to point in any desired direction on the map. The rotation angle is given relative to north on the map. For example, an angle of zero causes the pointer to point north and 90 degrees causes the pointer to point east.

9. When the map is rotated the pointer will also rotate to maintain its direction relative to the map.

10. There is an indicator visible on the screen that can display the values 0 through 360 degrees. This indicator can be set independently of the map orientation.

11. The device can be made to show a distinctive display, such as hash marks. This is known as "map warning".

12. The geographic position on the map under the reference point is available from this module. The values displayed by the map indicator, the map pointer angle, the map rotation angle, and the map scale are available from this module.

13. The map can be moved under the reference point without specifying a new geographic location to display. This is called slewing. The maximum slew rate is given by a system generation time parameter. The map cannot be slewed past an edge. If an edge is encountered then the map stops until it is moved in a direction away from the edge.

14. There are characteristics that define the different maps that may be used with the PMDS. These include area of the world covered, the scale of the map, and the dimensions of the map. An implementation for the PMDS virtual interface must be parameterized so that it can be reconfigured to function with any map that falls within a certain class of maps without program reassembly or reloading. Although the PMDS module must provide access functions to allow these parameters to be set, the parameters are not visible in the abstract interface description because they show the nature of the actual maps. The initialization functions are described in the PMDS reconfiguration section. Users of the other functions of the PMDS abstract interface must assume that the parameters have been set; if not, the virtual device will not operate correctly. See DI.11.3R.

DI.11.3.2. Assumptions about Undesired Events

1. User programs will not call +S_MAP_POSITION+ if the position is not displayable on the current map.

2. User programs will not call +NEXT_SCALE+ with parameter p1 equal to the grossest scale.

3. User programs will not call +PREV_SCALE+ with parameter p1 equal to the finest scale

DI.11.4. Interface Design Issues

1. In earlier versions the hardware decenter switch and its interpretation was a part of this module. This has been changed to allow the user of this module to set the map reference point in other ways.

2. We include only two of the possible reference points because we think it unlikely that others will be found useful.

DI.11.5. Local Types

Type name	Type definition
map_scale	The values of the map scale data type have a complete ordering from finest to grossest, where a map at scale finest shows the most detail and at scale grossest shows the largest area.
ref_pt	enumerated: $center$, $bottom-center$

DI.11.6. Local Dictionary

!+Map displayable+!	True iff the requested location can be displayed at the current scale. False otherwise.
!+Map indicator+!	The angle currently being displayed by the map indicator.
!+Map pointer angle+!	The current angle of the software controlled pointer on the map screen.
!+Map position+!	The geographic position under the map reference point on the screen.
!+Map position valid+!	True if a map valid map display is under the reference point (i.e. !+Map position+! is valid). False indicates that the map warning area is visible under the reference point.
!+Map reference pt+!	The current reference point on the map screen.
!+Map rotation+!	The current rotation angle of the map.
!+Map scale+!	The current scale that the map is operating with.

DI.11.7. Undesired Event Dictionary

%Not displayable%	+S_MAP_POSITION+ was called with a position that cannot be displayed at the current scale.
%Already grossest%	+NEXT_SCALE+ was called with p1 equal to the grossest scale.
%Already finest%	+PREV_SCALE+ was called with p1 equal to the finest scale.

DI.11.8. Function Effects

+DISPLAY_MAP_WARNING+	After being called the map screen shows a distinctive display, such as hashmarks, until another display is caused by +S_MAP_POSITION+.
+NEXT_SCALE+	p1 is set to the scale that is just grosser than its value at the time of the call
+PREV_SCALE+	p1 is set to the scale that is just finer than its value at the time of the call
+S_MAP_INDICATOR+	The indicator on the map screen is set to indicate the value of p1.
+S_MAP_POINTER_ANGLE+	The pointer is deflected to indicate the angle relative to map directions. The angle is measured clockwise from north on the map as seen by the pilot.
+S_MAP_POSITION+	If the last call to +G_MAP_DISPLAYABLE+ returned p3 with true then a map with the geographic location given by the parameters on that call is shown with the specified location under the reference point. If p3 is false then +DISPLAY_MAP_WARNING+ is invoked and the UE %Not Displayable% occurs.
+S_MAP_REFERENCE_PT+	The map reference point is the point on the map display screen which the requested location on the map is positioned under. $center$ sets the reference point to the center of the screen. $bottom-center$ sets the reference point to the bottom center edge of the screen. The change in reference point is immediate; it does not wait for the next +S_MAP_POSITION+ to become effective.
+S_MAP_ROTATION+	The map is rotated by angle p1. The rotation is given by the angle between the line from the center of the display to north on the map, and the line from the center of the display screen to the top-center of the display.
+S_MAP_SCALE+	The scale of the map is set to the value of p1. Future calls to +S_MAP_POSITION+ and +G_MAP DISPLAYABLE+ will use this scale value. The current position and scale of the map is not affected (until the next +S_MAP_POSITION+).

+SLEW_MAP+	The map is moved under the reference point vertically (as seen by the pilot) at a rate of p1 * ¢map_slew_rate¢. The map moves up when p1 is positive. The map is moved horizontally a. a rate of p2 * ¢map_slew_rate¢. The map moves to the right (as seen by the pilot) when p2 is positive. The maximum slew rate in either direction is ¢map_slew_rate¢ so values of p1 and p2 greater than 1 have the same effect as 1.

DI.11.9. System Generation Parameters

¢Map_lat_res¢	The resolution of the latitude parameter to position the map.
¢Map_indicator_res¢	The resolution of the indicator on the map screen.
¢Map_long_res¢	The resolution of the longitude parameter to position the map.
¢Num_map_scales¢	Number of map scales available.
¢Map_pointer_res¢	The resolution of the map pointer angle.
¢Map_rot_res¢	The re·olution of the map rotation angle.
¢Map_scale_array¢	Array of map scale values, number of entries equals ¢Num_map_scales¢. ¢Map_scale_array¢(1) is the finest map scale and ¢map_scale_array¢(¢Num_map_scales¢) is the grossest map scale.
¢Map_slew_rate¢	The maximum slew rate of the map.

DI.11.10. Information Hidden

1. This module hides the actual layout of the maps on the film cassettes, the number of different maps on one cassette, and the scales of the maps.

2. The operations necessary to cause the PMDS device to move the maps. The conversion to the map format from longitude and latitude is hidden.

3. The way in which the map reference point is stored and used.

DI.11.11. Calculations Performed Within the Module

It is necessary to compute from longitude and latitude a position on the map film strip and to compute the bounds of the displayable area for each available scale.

DI.11.12. Implementation Notes

1. The functions that return the number of scales, the finest scale value, and the grossest scale value can be implemented to function at assembly time. The functions that return the next and previous scale values of a variable will probably be used at run-time.

DI.11R. PROJECTED MAP DISPLAY SET RECONFIGURATION

DI.11.1R Introduction

The filmstrips loaded in the PMDS in a particular aircraft differ in ways that must be taken into account by the PMDS device interface module. These differences characterize the maps that can be projected on the screen. The PMDS device interface module is parameterized so that the maps can be changed without program reassembly or reloading. This section describes the access functions that allow these parameters to be read and written by other parts of the software. Since these functions reveal information about the maps, they are not included in the PMDS device interface description (DI.11). Only a small set of other programs need them; programmers of the rest of the system need not know that they exist.

This description is not meant to stand alone. It uses terms and ideas defined in the PMDS device interface description.

DI.11.2R. Interface Overview

DI.11.2.1R. Access Function Table

Function name	Parm type	Parm info	UEs
+G/S_MAP_LONGITUDE+	p1:mapid;I		
	p2:longitude;O/I	!+central long.+!	None
+G/S_MAP_LATITUDE_CT+	p1:mapid;I		
	p2:integer;O/I	!+low lat ct+!	
+G/S_MAP_ORIENTATION+	p1:mapid;I		
	p2:angle;O/I	!+map orient+!	

DI.11.3R. Basic Assumptions

1. There may be more than one map available in the PMDS. In the actual PMDS in the aircraft now, there are assumed to be two, called "A" and "B". These names are used by maintenance personnel who key in values characterizing the map to the software.

2. Each map is characterized by three pieces of information:

   the central longitude, given as an angle
   the bottom latitude, given as counts (see ref(1), section 4.5.5)
   the orientation of the map along the long axis of the map

   Unless the PMDS module is given this information, it cannot successfully position the map to display a given location.

DI.11.4R. Design Issues

1. We considered including these functions in the interface for the PMDS, but felt that they revealed too much information about the map filmstrips. As it is, only the programs that allow these parameters to be entered from the panel or displayed on the panel need to have access to these functions.

2. One of the parameters that must be entered is the lowest latitude of the map. We have chosen to enter an integer representation of the latitude into the module because that is the way that the parameter is given on the filmstrip and entered by the operator. Since the reconfiguration section is already device dependent nothing is gained by hiding this representation.

DI.11.5R. Local Types:

mapid	enumerated: A, B

DI.11.6R. Local Dictionary

!+central long.+!	The longitude of the central meridian of the map; positive for east, negative for west.
!+low lat ct+!	A signed number representing the southern most latitude of the map area covered; this number is in counts, where each count represents 8/9 of a degree from the equator. The number is positive for a north latitude; negative for a south latitude.
!+map orient+!	The angle between a meridian line on the map and the longitudinal axis of the filmstrip. Map orientation is zero degrees for north/south maps, 90 degrees for east/west maps.

DI.11.7R. Undesired Event Dictionary: none

DI.11.8R. Function Effects

Directly visible effects: The functions in this interface come in "G_"/"S_" pairs. After a call to "S_" function with p2 = n, calls to corresponding "G_" function return the value n.

Additional effects: Unless the "S_" functions have been called with the correct values for the actual filmstrips in the aircraft, calls to +S_MAP POSITION+ will not have the expected effect.

+S_FILM_LONGITUDE+	PMDS module assumes that the longitude of a line down the center of the map specified by p1 is p2
+S_FILM_LATITUDE_CT+	PMDS module assumes that the lowest latitude of the of the map specified by p1 is p2
+S_FILM_ORIENTATION+	PMDS module assumes that the map specified by p1 is oriented with at angle p2

DI.11.9R. System Generation Parameters: none

DI.11.10R. Information hidden: see DI.11

DI.11.11R. Calculations performed within the module: see DI.11

DI.11.12R. Implementation notes: see DI.11

DI.12. RADAR ALTIMETER

DI.12.1. Introduction

The Radar Altimeter is a sensor that measures the altitude of the aircraft above the local terrain, either land or water.

DI.12.2. Interface Overview

DI.12.2.1 Access Function Table

Function name	Parm type	Parm info	Calling UEs
+G_RADAR_ALT+	p1:distance;O	!+alt RADAR+!	None
	p2:logical;O	!+RA valid+!	

DI.12.2.2 Events Signalled

@T(!+RA valid+!) @F(!+RA valid+!)

DI.12.3. Basic Assumptions

1. The device is capable of measuring the altitude of the aircraft above the terrain.

2. The minimum and maximum altitude and the resolution of the measurement are given as system generation parameters and will not change without reassembling and reloading the software.

3. The altitude data received from this device may not be valid. A validity indicator is available with the data and an event is signalled when the validity of the most recent data is different than the previous data.

DI.12.4. Interface Design Issues

1. Should we design the interface on the assumption that an event will be signalled when the validity changed whether or not the data is being used? We could implement this by having the actual device read data constantly whether the user programs want it or not. We assumed that there would be little interest in this event when the device was not being used and decided for efficiency's sake not to check except when data was requested.

DI.12.5. Local Types: None

DI.12.6. Local Dictionary

!+alt RADAR+!	Aircraft altitude above the terrain as measured by the radar altimeter.
!+RA valid+!	true if most recent !+alt RADAR+! is valid; false if !+alt RADAR+! is invalid

DI.12.7 Undesired Event Dictionary: None

DI.12.8 Function Effects: None

DI.12.9 System Generation Parameters

¢Radar_alt_max¢	Maximum measurable altitude with this device.
¢Radar_alt_min¢	Minimum measurable altitude with this device.
¢Radar_alt_res¢	Resolution of radar altitude measurements.

DI.12.10. Information Hidden

1. The value encoding of the data words from the device.

2. Corrections that are applied to the data by this module.

3. How invalid data is identified.

DI.12.11. Calculations Performed Within the Module

Calculations performed within this module include decoding the data word and corrections required to the data.

DI.12.12. Implementation Notes: None

DI.13. SHIPBOARD INERTIAL NAVIGATION SYSTEM (SINS)

DI.13.1. Introduction

The shipboard inertial navigation system is a sensor that reports the position, velocity, and attitude of a ship selected by the aircraft. Reference 1 defines the coordinate system assumed by SINS device.

DI.13.2. Interface Overview

DI.13.2.1 Access Fuction Table

Function name	Parm type	Parm info	Undesired events
+G_SINS_AGE+	p1:time;0	!+SINS attitude age+!	%SINS not enabled%
	p2:time:0	!+SINS position age+!	
	p3:time:0	!+SINS velocity age+!	
+G_SINS_EAST_VEL+	p1:speed;0	!+SINS east vel+!	
+G_SINS_HEADING+	p1:angle;0	!+SINS heading+!	
+G_SINS_NORTH_VEL+	p1:speed;0	!+SINS north vel+!	
+G_SINS_PITCH+	p1:angle;0	!+SINS pitch+!	
+G_SINS_POSITION+	p1:longitude;0	!+SINS long+!	
	p2:latitude;0	!+SINS lat+!	
+G_SINS_ROLL+	p1:angle;0	!+SINS roll+!	
+G_SINS_VALIDITY+	p1:logical;0	!+SINS attitude valid+!	
	p2:logical;0	!+SINS position valid+!	
	p3:logical;0	!+SINS velocity valid+!	
+STOP_SINS+			
+START_SINS+			%SINS enabled%

DI.13.2.2 Events Signalled

@T(!+SINS attitude valid+!)	@F(!+SINS attitude valid+!)
@T(!+SINS position valid+!)	@F(!+SINS position valid+!)
@T(!+SINS velocity valid+!)	@F(!+SINS velocity valid+!)

DI.13.3.1. Basic Assumptions

1. The SINS provides three catagories of data about the situation of the ship sending the data: position, attitude, and velocity. Position data consist of the longitude and latitude of the ship. Attitude refers to the roll, true heading, and pitch of the ship. Velocity information is given as the magnitude of the ship's velocity in the east and north directions.

2. Valid SINS data might not be available at all times. Since the maximum rate that the ship's situation (position, attitude, or velocity) can change is known, a particular item of SINS data is valid for some amount of time before new data is required. If new data is not received within

the maximum allowed time, then an event is signalled to indicate that SINS data of that catagory is invalid. The last valid measurement is still available from the SINS module. When new valid SINS data is available then another event is signalled to indicate that valid data exist. It is the responsibility of the using programs to decide whether to use the data.

3. The elapsed time since the last valid data was received is available from the module.

4. The SINS device interface module can be turned on and off by the software. Computer time is conserved if the module is turned off when SINS data is not needed.

5. SINS attitude measurements are given in the coordinate system defined in appendix 3.

6. The range of values measurable by SINS and their resolution are given by system generation time parameters.

DI.13.3.2. Assumptions about Undesired Events

1. User programs will not call any access function of this module except +START_SINS+ if the module is not enabled.

2. User programs will not call +START_SINS+ if the module is already enabled.

DI.13.4. Interface Design Issues: None

DI.13.5. Local Data Types: None

DI.13.6. Local Dictionary

!+SINS attitude age+!	The elapsed time since new valid attitude data was provided by the SINS hardware.
!+SINS attitude valid+!	True iff SINS attitude data is valid.
!+SINS east vel+!	The east component of the ship's velocity. The value is positive in the positive Xs direction.
!+SINS heading+!	The angle measured from the line from the earth's north to the ship and the Ys axis looking down onto the ship from above.
!+SINS lat+!	The latitude of the ship as indicated by the SINS data.

!+SINS long+!	The longitude of the ship as indicated by the SINS data.
!+SINS north vel+!	The north component of the ship's velocity as indicated by the SINS data. The value is positive in the positive Ys direction.
!+SINS pitch+!	The angle between the ship's Ys axis and local earth horizontal plane. The angle is positive when the bow of the ship is above the horizontal plane.
!+SINS position age+!	The elapsed time since new valid position data was provided by the SINS hardware.
!+SINS position valid+!	<u>True</u> iff SINS position data is valid.
!+SINS roll+!	The angle between the ship's Xs axis and local earth horizontal plane. The angle is positive when the starboard side of the ship is down.
!+SINS velocity age+!	The elapsed time since new valid velocity data was provided by the SINS hardware.
!+SINS velocity valid+!	<u>True</u> iff SINS velocity data is valid.

DI.13.7. Undesired Event Dictionary

%SINS enabled%	<u>enabled</u> by +START_SINS+ <u>inhibited</u> by +STOP_SINS+
%SINS not enabled%	<u>enabled</u> when @T(power up) <u>enabled</u> by +STOP_SINS+ <u>inhibited</u> by +START_SINS+

DI.13.8. Function Effects

+START SINS+	This function enables the SINS device interface and permits the "G" functions to operate.
+STOP_SINS+	This function disables the SINS interface and the "G" functions.

DI.13.9. System Generation Parameters

¢SINS_attitude_res¢	The resolution of the SINS attitude measurements.
¢SINS_lat_res¢	The resolution of SINS latitude measurements.
¢SINS_long_res¢	The resolution of SINS longitude measurement.
¢SINS_pitch_max¢	The maximum magnitude of the pitch angle measurable by this SINS.
¢SINS_roll_max¢	The maximum magnitude of the roll angle measurable by this SINS.
¢SINS_velocity_max¢	The maximum velocity magnitude measurable with this SINS.
¢SINS_velocity_res¢	The resolution of SINS velocity measurements.

DI.13.10. Information Hidden

1. The representation of the SINS data within the I/O words, including scale, offset, and label information.

2. For security purposes the representation of the SINS data is classified. The interface provides unencoded data, and the classified information is used within this module only.

3. The method used to determine if SINS data in each catagory is valid.

4. The maximum age that the data may reach and still be considered valid.

DI.13.11. Calculations Performed Within the Module: None

DI.13.12. Implementation Notes

1. The process behind this interface will receive SINS data words, identify them, and determine their validity. Valid data items will be kept and supplied upon demand. Invalid data items are discarded.

2. The implementation of this module should be classified "CONFIDENTIAL".

DI.14. SLEW CONTROL INTERFACE

DI.14.1. Introduction

The slew control is a data entry device that can be used to enter a two-dimensional displacement from an origin.

DI.14.2. Interface Overview

DI.14.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G_SLEW_RIGHT_LEFT+	p1:real;O	!+Slew right-left+!	None
+G_SLEW_UP_DOWN+	p1:real;O	!+Slew up-down+!	

DI.14.2.2 Events Signalled

@T(!+Slew Displacement non-zero+!) @F(!+Slew Displacement non-zero+!)

DI.14.3. Basic Assumptions

1. There is a device that allows the operator to specify a two dimensional displacement from an origin. Displacement is measured along two orthogonal axes, called up-down and right-left. An example of this type of device is a 2-degree of freedom joystick.

2. The slew control is "up" if it is behind the center position and "down" if it is forward of its center position as viewed by the pilot. The slew control is to the right if the control is right of center and to the left if the control is left of center as viewed by the pilot.

3. The range and resolution of the values produced by the slew control are given as system generation parameters and will not vary without program reassembly and reloading.

4. Displacement values less than ¢Slew_min¢ are neglected and zero is returned.

DI.14.4. Interface Design Issues: None

DI.14.5. Local Types: None

DI.14.6. Local Dictionary

!+Slew right-left+!	This parameter indicates the right-left (as seen by the pilot) position of the slew control. If the control is in the center position this value is zero. A positive value indicates that the control is right of center and a negative value indicates left of center. The extreme values of the range indicate that the control is at its limit of right-left movement.
!+Slew up-down+!	This parameter indicates the up-down (seen by the pilot as toward him for up and away from him for down) position of the slew control. If the control is in its center position this value is zero. A positive value indicates that the control is above center and a negative value indicates below center. The extreme values of the range indicate that the control is at its limit of up-down movement.
!+Slew displacement non-zero+!	True iff !+Slew right-left+! neq 0 OR !+Slew up-down+! neq 0

DI.14.7. Undesired Event Dictionary: None

DI.14.8. Function Effects: None

DI.14.9. System Generation Parameters

¢Slew_min¢	The minimum usable value for slew control displacement along either axis. Values from the actual device below this are assumed to be noise and zero will be returned by this module to user programs.
¢Slew_up_max¢	Maximum value for the slew control displacement in the up-down direction. This is the value returned when the control is at an extreme of its movement along the up-down axis.
¢Slew_rl_max¢	Maximum value for the slew control displacement in the right-left direction. This is the value returned when the control is at an extreme of its movement along the right-left axis.
¢Slew_res¢	The resolution of the slew control outputs.

DI.14.10. Information Hidden

1. The value encoding of the data items used by these access functions.

2. The fact that the slew control is not a "perfect" device. It does not return exactly zero when in the center position. If the value from the slew control is less than a certain value then it is considered to be centered. The interface will return zero when the slew control position value is less than this minimum.

DI.14.11. Calculations Performed Within the Module

The only calculations performed within this module are those required to determine if the values from the device are above or below the noise level.

DI.14.12. Implementation Notes: None

DI.15. SWITCH BANK INTERFACE

DI.15.1. Introduction

The switch bank is a data entry device consisting of a set of switches. The switches fall into two classes, depending on the number of states: toggles with two states and selectors with more than two states. The switches and switch positions are named in accordance with the nomenclature in reference (1).

DI.15.2. Interface Overview

DI.15.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired events
+G_AUTOCAL_TOGGLE+	p1:logical;O	!+Auto-cal sw+!	None
+G_FLY_TO_NUM_SELECTOR+	p1:integer;O	!+Fly to num+!	
+G_FLY_TO_SELECTOR+	p1:fly_to_state;O	!+Fly to state+!	
+G_MAP_DECENTER_TOGGLE+	p1:logical;O	!+Map decenter+!	
+G_MAP_HOLD_TOGGLE+	p1:logical;O	!+Map hold+!	
+G_MAP_LDG_TOGGLE+	p1:logical;O	!+Map ldg+!	
+G_MAP_NORTH_UP_TOGGLE+	p1:logical;O	!+Map north-up+!	
+G_MAP_SCALE_TOGGLE+	p1:logical;O	!+Map scale sw+!	
+G_PANEL_MODE_SELECTOR+	p1:panel_mode;O	!+Panel mode+!	
+G_PANEL_UPDATE_SELECTOR+	p1:update;O	!+Update+!	
+G_PRES_POSITION_SELECTOR+	p1:pp_mode;O	!+Pres pos+!	
+G_SELF_TEST_TOGGLE+	p1:logical;O	!+Self-test+!	

DI.15.2.2 Events Signalled

```
@T(!+Auto-cal changed+!)     @T(!+Fly to num changed+!)
@T(!+Fly to state changed+!) @T(!+Map decenter changed+!)
@T(!+Map hold changed+!)     @T(!+Map ldg changed+!)
@T(!+Map north-up changed+!) @T(!+Map scale sw changed+!)
@T(!+Panel mode changed+!)   @T(!+Pres pos changed+!)
@T(!+Self-test changed+!)    @T(!+Update changed+!)
```

DI.15.3. Basic Assumptions

1. There are a number of switches whose state the software can detect that have no effect on any devices other than the computer. There are two types of switches; toggles and selectors. Toggles return a logical value and selectors return one of several possible values.

2. The "fly-to number" selector has a number of possible settings and returns an integer representing its current setting. The integers range from 0 to ¢Fly_to_max¢, where ¢Fly_to_max¢ is the maximum possible setting. The integer returned is the same as the value on the selector that the operator can see. ¢Fly_to_max¢ is constant after program assembly time.

3. The switch accessed by +G_AUTOCAL_TOGGLE+ cannot be operated when the aircraft is airborne.

DI.15.4. Interface Design Issues

1. All of the switches described here have no direct effect on any devices. For example the auto-calibration switch is sensed only by the computer. The switch is named "AUTO-CAL" because of the way that the switch is labeled. The alternative would be to talk of switch-1, switch-2, ..., and switch-n. It is felt that such abstraction would be confusing and would not be worth the disadvantages.

2. Earlier we included the master function switch (MFS) in this interface. It has been removed. This was done because the MFS has direct effects on several devices other than the computer (in particular the FLR, ASCU, and HUD). All switches described in this interface could have different functions with only software changes.

DI.15.5. Local Types

Type name	Type definition
fly_to_state	enumerated: $Dest$ $Mark$
panel_mode	enumerated: $None$, $Prespos$, $Dest$, $Mark$, Rng/Brg, $DBHT$, $ALTMSLP$
pp_mode	enumerated: $LatLong$, $Update$, $Wind$
update	enumerated: $Data$, HUD, $Radar$, $Flyover$, $Loran$, $TacL-L$, $Tacan$, $IMS-HUD$, $SINSX-Y$, $Z-DHDG$

DI.15.6. Local Dictionary

!+Auto-cal sw+!	True if switch labeled "Auto-Cal" set to on position
!+Auto-cal changed+!	True while !+Auto-cal+! is changing value.

!+Fly to num+!	The setting of the numeric selector labeled "Fly to".
!+Fly to num changed+!	True while !+Fly to num+! is changing value.
!+Fly to state+!	The setting of the selector labeled "Fly to".
!+Fly to state changed+!	True while !+Fly to state+! is changing value.
!+Map hold+!	True if switch labeled "Hold" on PMDS set to on position
!+Map hold changed+!	True while !+Map hold+! is changing value.
!+Map decenter+!	True iff switch labeled "Decenter" on PMDS set to on postition
!+Map decenter changed+!	True while !+Map decenter+! is changing value.
!+Map ldg+!	True iff switch labeled "Map landing" on PMDS set to on position.
!+Map ldg changed+!	True while !+Map ldg+! is changing value.
!+Map north-up+!	True if switch labeled "North-Up" on PMDS set to on position
!+Map north-up changed+!	True while !+Map north-up+! is changing value.
!+Map scale sw+!	True if toggle set to "80"
!+Map scale changed+!	True while !+Map scale+! is changing value.
!+Panel mode+!	The setting of the panel mode selector switch. Values correspond to switch nomemclature.
!+Panel mode changed+!	True while !+Panel mode+! is changing value.
!+Pres pos+!	The setting of the present position selector switch. Values correspond to switch nomemclature.
!+Pres pos changed+!	True while !+Pres pos+! is changing value.
!+Self-test+!	True if switch labeled "Test" on panel set to on position.
!+Self-test changed+!	True while !+Self-test+! is changing value.
!+Update+!	The setting of the update selector switch. Values correspond to switch nomenclature.
!+Update changed+!	True while !+Update+! is changing.

DI.15.7. Undesired Event Dictionary: None

DI.15.8. Function Effects: None

DI.15.9. System Generation Parameters

¢Fly_to_max¢ Maximum number of settings for the fly-to number selector. The selector returns 0 through ¢FLY_TO_MAX¢.

DI.15.10. Information Hidden

1. The value encoding of the data items used by these access functions.

2. The maximum number of settings for the fly-to number selector (before program assembly time). The maximum number of switch positions is known to the program at run time.

DI.15.11. Calculations Performed Within the Module: None

DI.15.12. Implementation Notes: None

DI.16. TACTICAL AIR NAVIGATION SYSTEM (TACAN)

DI.16.1. Introduction

The TACAN system is a sensor that measures bearing and slant range from the aircraft to a pilot-selected TACAN station. The geographic location of the TACAN station being received must be known to the calling programs in order for the bearing and range data to be meaningful.

DI.16.2. Interface Overview

DI.16.2.1 Access Functions Table

Function name	Parm type	Parm info	Undesired events
+G_TACAN_BEARING+	p1:angle;O	!+TAC bearing+!	None
	p2:logical;O	!+TAC bearing valid+!	
+G_TACAN_RANGE+	p1:distance;O	!+TAC range+!	
	p2:logical;O	!+TAC range valid+!	

DI.16.2.2 Events Signalled

@T(!+TAC bearing valid+!)	@F(!+TAC bearing valid+!)
@T(!+TAC range valid+!)	@F(!+TAC range valid+!)

DI.16.3. Basic Assumptions

1. The TACAN system is capable of measuring the bearing angle from the aircraft to the TACAN station currently being received. TACAN bearing is the angle measured CW looking down onto the aircraft between the line from the aircraft to magnetic north and the line from the aircraft to the selected TACAN station.

2. The TACAN system is capable of the measuring slant range from the aircraft to the TACAN station being received. Slant range is the straight line distance from the aircraft to the TACAN station (not the range along the ground).

3. At some times valid TACAN data is not available. After an attempt to read data the software can determine if TACAN data is valid. An event is signalled and the last valid TACAN data is provided when no new data is available. Another event is signalled when new valid data has been read from the module.

DI.16.4. Interface Design Issues: None

DI.16.5. Local Data Types: None

DI.16.6. Local Dictionary

!+TAC bearing+!	The angle measured CW looking down onto the aircraft between the line from the aircraft to magnetic north and the line from the aircraft to the selected TACAN station. This is a stale value if !+TAC bearing valid+! is false.
!+TAC bearing valid+!	True iff !+TAC bearing+! from the same function call is valid.
!+TAC range+!	The straight line distance from the aircraft to the current TACAN station. This is a stale value if !+TAC range valid+! is false.
!+TAC range valid+!	True iff !+TAC range+! from the same function call is valid.

DI.16.7. Undesired Event Dictionary: None

DI.16.8. Function Effects: None

DI.16.9. System Generation Parameters

¢TACAN_bearing_res¢	The resolution of the TACAN bearing measurement.
¢TACAN_range_max¢	The maximum range that can be measured with the TACAN system.
¢TACAN_range_res¢	The resolution of the TACAN range measurement.

DI.16.10. Information Hidden

1. The value encoding of the data from the TACAN system.
2. The conditions that determine whether the TACAN data is valid.

DI.16.11. Calculations Performed Within the Module: None

DI.16.12. Implementation Notes

The current TACAN receiver gives no indication to the software if good TACAN data are available. The software must read the data, then check the validity bit. Therefore these access functions simply read the data items and provide the validity indicator and the data.

DI.17. VISUAL INDICATORS (Auto-Cal Indicator and Non-Align Indicator)

DI.17.1. Introduction

There are two visual indicators controlled by the OFP on the A-7 aircraft; one that can, and one that cannot, be seen by the pilot during flight. These are currently labeled "IMS Non-aligned" and "Auto-Cal", respectively. Each can be on steady, on blinking, or off.

DI.17.2. Interface Overview

DI.17.2.1. Access Function Table

Function name	Parm type	Parm Info	Undesired events
+G/S_AUTOCAL_INDICATOR+	p1:ind_cntrl;O/I	!+Auto-cal+!	None
+S_AUTOCAL_BLINK_RATE+	p1:real;I	blink/sec	
+G/S_NON_ALIGN_INDICATOR+	p1:ind_cntrl;O/I	!+Non-align+!	
+S_NON_ALIGN_BLINK_RATE+	p1:real;I	blink/sec	

DI.17.3. Basic Assumptions

1. There are two visual indicators that can be controlled by the software. The indicators can be off, on steady, or blinking on and off. One of these, labeled "Auto-Cal" is not visible by the pilot during flight. The other, labled "IMS non-align" is visible by the pilot during flight.

2. The "Auto-Cal" indicator can be seen by the person operating the Auto-Cal switch (reference (2)).

3. The blinking rate is variable and can be set by the calling program, but will default to a system generation time parameter until the set rate function is called.

4. The state of the indicators is available from this module.

DI.17.4. Interface Design Issues

The indicators are referred to here as they are labeled ("IMS Non-align", "Auto-Cal", "Computer Fail") rather than in a more abstract way. The mnemonic value outweighs the expected cost of change savings; the change is unlikely and the cost is only the name in the documents.

DI.17.5. Local Types

Type Name	Type Definition
ind_cntrl	enumerated: \$On\$, \$Intermittent\$, \$Off\$

DI.17.6. Local Dictionary

!+Auto-cal+!	The state of the auto-cal indicator as last set by +S_AUTOCAL_INDICATOR+.
!+Non-align+!	The state of the non-align indicator as last set by +S_NON_ALIGN_INDICATOR+

DI.17.7. Undesired Event Dictionary: None

DI.17.8. Function Effects

+S_AUTOCAL_INDICATOR+	*if* p1=On	*then* "Auto-Cal" indicator turned on
	if p1=Off	*then* "Auto-Cal" indicator turned off
	if p1=$Intermittent$	*then* "Auto-Cal" indicator turned on and off at rate set by +S_AUTOCAL_BLINK_RATE+
+S_AUTOCAL_BLINK_RATE+	When commanded to blink the "Auto-Cal" indicator will blink at p1/sec.	
+S_NON_ALIGN_INDICATOR+	*if* p1=On	*then* "Non-Align" indicator turned on
	if p1=Off	*then* "Non-Align" indicator turned off
	if p1=$Intermittent$	*then* "Non-Align" indicator turned on and off at rate set by +S_NON_ALIGN_BLINK_RATE+
+S_NON_ALIGN_BLINK_RATE+	When commanded to blink the "Non-align" indicator will blink at p1/sec.	

DI.17.9. System Generation Parameters

¢Auto-Cal blink default¢	Default blink rate for "Auto-Cal" indicator.
¢Non-Align blink default¢	Default blink rate for "Non-Align" indicator.

DI.17.10. Information Hidden

1. The value encoding of the data words to the devices.

DI.17.11. Calculations Performed Within the Module: None

DI.17.12. Implementation Notes: None

DI.18 WAYPOINT INFORMATION SYSTEM (WIS)

DI.18.1. Introduction

The waypoint information system provides the longitude and latitude of a position that is transmitted to the aircraft from an external source.

DI.18.2. Interface Overview

DI.18.2.1. Access Function Table

Function name	Parm type	Parm info	Undesired events
+G_WAYPT_POSITION+	p1:latitude;O	!+WAYPT lat+!	None
	p2:longitude;O	!+WAYPT long+!	
	p3:integer;O	!+WAYPT ID+	

DI.18.2.2. Events Signalled

@T(!+WAYPT available+!)

DI.18.3. Assumptions Lists

DI.18.3.1. Basic Assumptions

1. It is possible for an external source to transmit positional information to the aircraft.
2. The maximum value of the waypoint identifier and the reported position are system generation time parameters and are fixed at execution time.

DI.18.4. Interface Design Issues: None.

DI.18.5. Local Types: None.

DI.18.6. Local Dictionary

!+WAYPT lat+!	The received waypoint latitude.
!+WAYPT long+!	The received waypoint longitude.
!+WAYPT ID+!	The identification number of the received waypoint position.
!+WAYPT available+!	True when new waypoint data is available. Set to false when data +G_WAYPT_POSITION+ is called.

DI.18.7. Undesired Event Dictionary: None.

DI.18.8. Function Effects: None.

DI.18.9. System Generation Parameters

¢WAYPT_lat_res¢	The resolution of waypoint latitude values.
¢WAYPT_long_res¢	The resolution of waypoint longitude values.
¢WAYPT_id_max¢	The maximum value of the waypoint identifier.

DI.18.10. Information Hidden

1. The format of received data (this format is CONFIDENTIAL for security reasons).
2. The details of the waypoint I/O device.

DI.18.11 Calculations Performed Within the Module: None.

DI.18.12 Implementation Notes

1. The implementation should be classified CONFIDENTIAL.

DI.19. WEAPON CHARACTERISTICS MODULE (WCM)

DI.19.1. Introduction

The Weapon Characteristics Module (WCM) provides weapon-dependent characteristics about the weapon on the currently active weapon stations (see Weapon Release System). The weapon-dependent characteristics provided by the WCS include a weapon class affecting the choice of ballistics equations, weapon measurements, minimum release interval, and release pulse width.

DI.19.2. Interface Overview

DI.19.2.1 Access Function Table

Function name	Param Type	Parm Info	Undesired events
+G_WEAPON_CLASS+	p1:weap_class;O	!+Weapon Class+!	%None%
+G_A1+	p1:real;O	!+A1+!	%Wrong class%
+G_A2+	p1:real;O	!+A2+!	
+G_A3+	p1:real;O	!+A3+!	
+G_A4+	p1:real;O	!+A4+!	
+G_A5+	p1:real;O	!+A5+!	
+G_A6+	p1:real;O	!+A6+!	
+G_A7+	p1:real;O	!+A7+!	
+G_BORESIGHT_ANGLE+	p1:angle;O	!+boresight angle+!	
+G_B1+	p1:real;O	!+B1+!	
+G_CR2+	p1:real;O	!+CR2+!	
+G_CR3+	p1:real;O	!+CR3+!	
+G_CR4+	p1:real;O	!+CR4+!	
+G_CR5+	p1:real;O	!+CR5+!	
+G_CR6+	p1:real;O	!+CR6+!	
+G_CR7+	p1:real;O	!+CR7+!	
+G_CR8+	p1:real;O	!+CR8+!	
+G_CR9+	p1:real;O	!+CR9+!	
+G_CR10+	p1:real;O	!+CR10+!	
+G_CR11+	p1:real;O	!+CR11+!	
+G_CR12+	p1:real;O	!+CR12+!	
+G_CR13+	p1:real;O	!+CR13+!	
+G_FLARE_FLAG+	p1:logical;O	!+Flare flag+!	
+G_HMAX+	p1:real;O	!+HMax+!	
+G_HMIN+	p1:real;O	!+HMin+!	
+G_LAT_EJECTION_SPEED+	p1:speed;O	!+lateral eject speed+!	
+G_LAUNCH_SPEED+	p1:speed;O	!+launch speed+!	
+G_LONG_EJECTION_SPEED+	p1:speed;O	!+longitudinal eject speed+!	
+G_MAGNUS_DEFLECTION+	p1:angle;O	!+horiz magnus deflection+!	
+G_MAX_RANGE+	p1:distance;O	!+max range+!	
+G_MAX_USEFUL_RANGE+	p1:distance;O	!+max useful range+!	
+G_MINE_FLAG+	p1:logical;O	!+Mine flag+!	
+G_MOTOR_BURN_TIME+	p1:time;O	!+motor burn time+!	
+G_MRI_CLASS+	p1:mri_class;O	!+MRI class+!	
+G_MUZZLE_VEL+	p1:speed;O	!+muzzle velocity+!	
+G_RACK_DELAY+	p1:time;O	!+rack delay time+!	
+G_RC_CLASS+	p1:rc_class;O	!+rack curve class+!	
+G_RELEASE_PULSE_WIDTH+	p1:time;O	!+release pulse width+!	
+G_U+	p1:speed;O	!+term vel sea level+!	
+G_U_CORRECTED+	p1:speed;O	!+term vel burst height+!	
+G_WCDA+	p1:real;O	!+WCDA+!	
+G_WEAPON_WARMUP_TIME+	p1:time;O	!+warmup time+!	

DI.19.2.2. Events Signalled: None.

DI.19.2.3 Access Functions and Weapon Classes

The following table shows the access functions legal for each of the possible weapon classes.

Weapon class	Functions legal in each mode
GN	+G_MUZZLE_VEL+, +G_CR2+, +G_BORESIGHT_ANGLE+, +G_CR12+ +G_CR13+, +G_MAGNUS_DEFLECTION+, +G_MAX_USEFUL_RANGE+ +G_MAX_RANGE+
HD	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_U+ +G_WCDA+, +G_A1+, +G_A2+
MF	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+ +G_MINE_FLAG+, G_FLARE_FLAG+
MD	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_A2+ +G_A1+, +G_WCDA+, +G_A4+, +G_A5+, +G_A6+, +G_A7+
OD	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+ +G_WCDA+, +G_A4+, +G_A5+, +G_A6+, +G_A7+, +G_A1+ +G_A2+, +G_B1+
OR	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_A2+ +G_A1+, +G_WCDA+, +G_A4+, +G_A5+, +G_A6+, +G_A7+
RK	+G_LAUNCH_SPEED+, +G_CR2+, +G_BORESIGHT_ANGLE+, +G_CR12+ +G_CR13+, +G_MOTOR_BURN_TIME+, +G_MAX_USEFUL_RANGE+ +G_MAX_RANGE+, +G_CR3+, +G_CR4+, +G_CR5+, +G_CR6+, +G_CR7+ +G_CR8+, +G_CR9+, +G_CR10+, +G_CR11+
SH	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_U+
SK	+G_RELEASE_PULSE_WIDTH+, +G_LAUNCH_SPEED+, +G_CR2+ +G_BORESIGHT_ANGLE+, +G_WEAPON_WARMUP_TIME+
SL	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_B1+
SM	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+ +G_A4+, +G_A5+, +G_A6+, +G_A7+, +G_U+

SOD	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+, +G_B1+ +G_U_CORRECTED+, +G_A2+, +G_A1+, +G_WCDA+, +G_A4+, +G_HMAX+ +G_HMIN+
SSH	+G_MRI_CLASS+, +G_RELEASE_PULSE_WIDTH+, +G_RC_CLASS+ +G_LONG_EJECTION_SPEED+, +G_LAT_EJECTION_SPEED+ +G_B1+, +G_U_CORRECTED+, +G_A2+, +G_A1+ +G_WCDA+, +G_A4+, +G_HMAX+, +G_HMIN+
UN	None
WL	+G_RELEASE_PULSE_WIDTH+

DI.19.3.1. Basic Assumptions

1. Every weapon type belongs to one and only one weapon class.

2. This module is capable of obtaining any information required to identify the weapon on the currently active weapon station(s).

3. The programs using this module will deal with only one weapon type at a time, and all programs using this module will know which weapon class is currently being considered.

4. The weapon dependent data provided by this module is dependent on the implementation of the weapon ballistic calculations. A change in the weapon equations may require a change in the information provided.

5. If no weapon stations are active then the weapon class returned by this module is UN.

DI.19.3.2. Assumptions about Undesired Events

1. User programs will not call access functions in violation of table DI.19.2.2. that relates functions with weapon classes.

DI.19.4. Interface Design Issues

1. In earlier versions of this and the Weapon Release System the weapon type code was provided by an get function from the WRS and input to the WCM. This had the disadvantage that the weapon type code was not being hidden and details of its meaning was required in two different modules. We have removed the weapon type from the WRS and no longer have a function to input it to the WCM. We now assume that the WCM will get whatever information it needs to identify the weapon on the currently active weapon station(s). This has the benefit of hiding the weapon type code, and the other items required to completely identify the current weapon.

2. Some of the weapon parameters have non-mnemonic names such as A4, CR13, etc. This is done here because these are the names used in reference (3) and it is desirable to keep the documents compatible. We expect to use the equations as presented in reference (3) without much modification.

DI.19.5. Local Data Types

mri_class	enumerated:	\$1\$, \$2\$, \$3\$
rc_class	enumerated:	\$a\$, \$b\$, \$c\$
weap_class	enumerated:	see section 19.2.2

DI.19.6. Local Dictionary

!+A1+! !+A2+! !+A3+!	Equation constant (page 24, reference 3).
!+A4+! !+A5+! !+A6+! !+A7+!	Equation constant (page 32, reference 3).
!+B1+!	Equation constant (page 15, reference 3).
!+boresight angle+!	The angle between the aircraft Ya axis and the weapon boresight in the Ya-Za plane. The angle is negative if the weapon boresight line points down when the aircraft is level.
!+CR2+! !+CR3+! !+CR4+! !+CR5+! !+CR6+! !+CR7+! !+CR8+! !+CR9+! !+CR10+! !+CR11+! !+CR12+! !+CR13+!	Equation constants relevant to guns and rockets (page 53, reference 3).
!+Flare flag+!	True iff weapon is a flare.
!+HMax+! !+HMin+!	Special weapon constant.
!+lat eject speed+!	The ejection speed of the weapon along the Xa axis.
!+launch speed+!	The launch speed of a rocket or missile.

!+long eject speed+!	The ejection speed of the weapon along the Ya axis.
!+magnus deflection+!	The change in the horizontal flight path of a bullet caused by a gyroscopic force exerted on it because of its spin and the air drag. The angle is measured from the boresight line to the true flight path of the bullet.
!+max range+!	The maximum range of the weapon under ideal conditions.
!+max useful range+!	The maximum range of the weapon under typical conditions.
!+Mine flag+!	True iff the current weapon is a mine.
!+motor burn time+!	The duration of powered flight for rockets and missiles.
!+MRI class+!	The minimum release interval class of the current weapon.
!+muzzle velocity+!	The muzzle velocity of the gun.
!+rack curve class+!	The identification of the proper curve to use to compute rack delays in some cases.
!+rack delay time+!	The delay that the ejector rack causes to the release pulse when used with the current weapon.
!+release pulse width+!	The release pulse width required by this weapon.
!+term vel burst height+!	The magnitude of the terminal velocity at burst height.
!+term vel sea level+!	The magnitude of the terminal velocity of the weapon at sea level.
!+warmup time+!	The warmup time required by this weapon.
!+WCDA+!	weight * cross sectional area of the weapon
!+Weapon Class+!	The class of the weapon loaded on the currently active weapon stations(s).

DI.19.7. Undesired Event Dictionary

%Wrong class%	An access function was called that is illegal for the current weapon class. See table 19.2.2.

DI.19.8. Function Effects: None.

DI.19.9. System Generation Parameters: None.

DI.19.10. Information Hidden

1. The way that the various weapon characteristics are produced is a secret of this module. For example they may be held in a table for some weapon types and produced from a model for others.

2. For some weapon types the setting of the "retarded" ASCU switch has an effect on the weapon characteristics. Which weapons it affects and the particular characteristics it affects are secrets of this module.

3. The information and the source of the information required to determine the identity of the weapon type on the currently active weapon station(s).

DI.19.11 Calculations Performed Within the Module: None

DI.19.12 Implementation Notes: None

DI.20. WEAPON RELEASE SYSTEM

DI.20.1. Introduction

The Weapon Release System is both a data entry device and a control device: it provides data about the weapon delivery options selected by the pilot, such as high drag, and it generates signals that cause weapons to be prepared and released. The Weapon Release System includes most functions of the Armament Release Panel (ARP) and the Armament Station Control Unit (ASCU).

The aircraft has a number of weapon stations attached under the wings. From the weapon stations, enabled by the pilot and loaded with legal weapons, the WRS selects one or two station to be the current active stations. One station is active if a pairs release is not selected and two stations are active if a pairs release is selected. Weapons are released from the active stations.

DI.20.2. Interface Overview

DI.20.2.1 Access Function Table

Function name	Parameter type	Parameter info	Undesired events
+G_ACTIVE_STATIONS+	p1:logical array;O	!+Active stations+!	None
+G_ARP_INTERVAL+	p1:integer;O	!+ARP Interval+!	
+G_ARP_PAIRS+	p1:logical;O	!+ARP Pairs+!	
+G_ARP_QUANTITY+	p1:integer;O	!+ARP Quantity+!	
+G_GUN_ENABLE+	p1:logical;O	!+Gun Enable+!	
+G_HIGH_DRAG+	p1:logical;O	!+High Drag+!	
+G_RDY_STATIONS+	p1:logical array;O	!+Ready Stations+!	
+G_MASTER_ARM+	p1:logical;O	!+Master Arm+!	
+G_REL_IN_PROGRESS+	p1:logical;O	!+Rel in Progress+!	
+G_WEAPON_MODE+	p1:weap_mode;O	!+Weapon Mode+!	
+G_MULT_RACK+	p1:logical;O	!+Mult Rack+!	%No active stations%
+RELEASE_WEAPON+ +WARMUP_WEAPON+	p1:time;I	release pulse width	%No active station% %Release in progress%

DI.20.2.2 Events Signalled

@T(!+ARP Interval changed+!)	@T(!+ARP Pairs changed+!)
@T(!+ARP Quantity changed+!)	@T(!+Gun Enable changed+!)
@T(!+High Drag changed+!)	@T(!+Master Arm changed+!)
@T(!+Weapon Mode changed+!)	@T(!+TD pressed+!)
@F(!+TD pressed+!)	@T(!+RE pressed+!)
@F(!+RE pressed+!)	

DI.20.3.1. Basic Assumptions

1. The setting of the following weapon related switches can be detected by the software: ARP Interval, ARP Pairs, ARP Quantity, Gun enable, Retarded Weapon (High Drag), Master Arm, Master Function Switch (Weapon Mode), Target Designate Button, and Release Enable Button. All of these switches have direct hardware effect on devices, therefore the assignment of functions is not up to the software designers.

2. There are a number of weapon stations controlled by this module. A station is ready if it has been selected by the pilot and is loaded with a legal weapon type known to the WRS. Any number of weapon stations (from none to all of them) may be ready at a time. This module will provide the identity of ready stations.

3. There is a priority ranking among the weapon stations, such that if more than one is ready, the highest priority station becomes the active station. Two stations become active if a pairs release has been selected. The active stations remain active until all of their weapons have been released or until they are deselected by the pilot (i.e. they become not ready). This module will provide the identity of active stations.

4. It is possible to release weapons either singly or in pairs.

5. Some weapon types require a warm-up before release. The warm-up weapon function must be called in time to allow the required interval to elapse before weapon release or the weapon may fail to perform properly. If the warmup time for a given weapon is zero then the warmup function need not be called at all.

DI.20.3.2. Assumptions about Undesired Events

1. User programs will not call +G_MULT_RACK+, +RELEASE_WEAPON+, or +WARMUP_WEAPON+ if there is no active weapon station.

2. User programs will not call +RELEASE_WEAPON+ or +WARMUP_WEAPON+ if a release is currently in progress.

DI.20.4. Interface Design Issues

1. Currently only the Shrike missile requires a warm-up time prior to release. Earlier versions of this interface (prior to version 8) had a function +PREPARE_SHRIKE+ to start the warm-up. This interface has been changed to make this feature more general. The required warm-up time is available from the weapon characteristics module and the using programs must call the function +WARMUP_WEAPON+ at that required time before release. This was done to hide the unique characteristic of the Shrike missile in the weapon characteristic module and to allow for the possibility that another weapon type might require this same type of warm-up.

2. Earlier we included the weapon characteristics (often called the ASCU table) in this module. We have decided to separate the weapon characteristics from the characteristics of the weapon release system (ASCU, ARP, PGS). This was done because changes in these two areas can be made independently of each other.

3. We have decided not to include a function to get the weapon type in this module (even though that infomation does come from the ASCU hardware). The weapon type code is required only by the Weapon Characteristics Module (WCM) and the weapon type code can be hidden completely by designing the WCM to get the weapon type directly.

DI.20.5. Local Types

Type name	Type definition
weap_mode	enumerated: $NATT$, BOC, $CCIP$, $NATTOFF$, $BOCOFF$, $None$

DI.20.6. Local Dictionary

!+Active Stations+!	This array is indexed by the weapon station numbers (from 1 to ¢NUM_WEAP_STATIONS¢). A value of <u>true</u> indicates that the corresponding station is an active station.
!+ARP Interval+!	The value set on the ARP switch labeled "Interval-Ft".
!+ARP Interval changed+!	<u>True</u> while !+ARP Interval+! is changing value.
!+ARP Pairs+!	<u>True</u> iff ARP switch labeled "Pair", "Single", "Simult Rkts" set to the "Pair" position.
!+ARP Pairs changed+!	<u>True</u> while !+ARP Pairs+! is changing value.
!+ARP Quantity+!	The value set on the ARP switch labled "Quantity".

!+ARP Quantity changed+!	True while !+ARP Quantity+! is changing value.
!+Gun Enable+!	True iff the gun is currently enabled.
!+Gun Enable changed+!	True while !+Gun Enable+! is changing value.
!+High Drag+!	True iff the switch labeled "RET WPN" is selected.
!+High Drag changed+!	True while !+High Drag+! is changing value.
!+Master Arm+!	True iff the master arm switch is selected.
!+Master Arm changed+!	True while !+Master Arm+! is changing value.
!+Mult Rack+!	True iff the active weapon station contain a multiple or triple ejector rack. Valid only if at least one element of !+Ready Stations+! = true!
!+Ready Stations+!	This array is indexed by the weapon station numbers (from 1 to ¢NUM_WEAP_STATIONS¢). A value of true indicates that the corresponding station is a ready station.
!+RE pressed+!	True iff the release enable button is pressed.
!+Rel In Progress+!	True if //BMBREL//=On.
!+TD pressed+!	True iff the target designate button is pressed.
!+Weapon Mode+!	The setting of the master function switch (weapon mode)
!+Weapon Mode changed+!	True while !+Weapon Mode+! is changing value.

DI.20.7. Undesired Event Dictionary

%No active stations%	revealed by +G_ACTIVE_STATIONS+ all elements of pl = false shows enabled at least one element of pl = true shows inhibited

%Release in progress%	revealed by +G_REL_IN_PROGRESS+
	pl=true shows enabled
	pl=false shows inhibited

DI.20.8. Function Effects

+WARMUP_WEAPON+	Initiates events to cause the warmup required by certain weapons.
+RELEASE_WEAPON+	Causes the fire ready and bomb release signals to the active weapon station. After time interval pl elapses both of these signals are turned off. The function returns control to the caller immediately after turning the signals on.

DI.20.9. System Generation Parameters

¢ARP_interval_max¢	The maximum setting of the interval switch.
¢ARP_quantity_max¢	The maximum setting of the quantity switch.
¢Num_weap_stations¢	The number of weapon stations controlled by this module. Note that the A/C may have some weapon stations that cannot be controlled by the software.

DI.20.10. Information Hidden

1. Details of the ARP, ASCU, and pilot's grip stick hardware, such as operations necessary to perform certain functions and the encoding of values.

2. The maximum settings of the ARP interval and quantity switch and the number of stations and weapon types are secrets at program design and write time. The actual values are inserted during program assembly.

3. The way that the ready and active weapon stations are identified.

4. The station priority scheme.

DI.20.11. Calculations Performed Within the Module

Other than scaling of input values no calculations are done in this module.

DI.20.12. Implementation Notes: None.

DI.21. WEIGHT ON GEAR SENSOR

DI.21.1. Introduction

The weight on gear device is a sensor that detects whether or not the aircraft is resting on its landing gear. This data can be used to infer whether or not the aircraft is airborne.

DI.21.2. Interface Overview

DI.21.2.1 Access Function Table

Function name	Parm type	Parm info	Undesired Events
+G_WEIGHT_ON_GEAR+	p1:logical;O	!+WOG+!	None

DI.21.3. Basic Assumptions

1. There is a sensor that indicates if the weight of the aircraft is resting on the landing gear.

DI.21.4. Interface Design Issues

1. The primary use of the weight on gear switch is to determine if the aircraft is airborne. The current OFP makes the assumption that if there is no weight on the switch then the aircraft is airborne. We have decided not to put that assumption into this interface. The conditions for determining if the aircraft is airborne will be left to the system status module. We may want to have a more complex condition such as one that makes use of altitude data as well as the weight on gear switch.

DI.21.5. Local Types: None

DI.21.6. Local Dictionary

!+WOG+!	True iff weight on landing gear detected

DI.21.7. Undesired Event Dictionary: None

DI.21.8. Function Effects: None

DI.21.9. System Generation Parameters: None

DI.21.10. Information Hidden

1. The value encoding of the data words from the devices.

DI.21.11. Calculations Performed Within the Module: None

DI.21.12. Implementation Notes: None

DI.A1 APPENDIX 1

ACCESS FUNCTION INDEX

Function Name	Device Interface	Section
+CLEAR_LOWER+	Panel	DI.10
+CLEAR_MARK+	Panel	DI.10
+CLEAR_UPPER+	Panel	DI.10
+DISPLAY_MAP_WARNING+	PMDS	DI.11
+G_A1+	WCM	DI.19
+G_A2+	WCM	DI.19
+G_A3+	WCM	DI.19
+G_A4+	WCM	DI.19
+G_A5+	WCM	DI.19
+G_A6+	WCM	DI.19
+G_A7+	WCM	DI.19
+G_ACTIVE_STATIONS+	WRS	DI.20
+G_ADC_ALTITUDE+	ADC	DI.1
+G_ADC_LPROBE+	ADCR	DI.1R
+G_ADC_MACH_INDEX+	ADC	DI.1
+G_ADC_TRUE_AIRSPEED+	ADC	DI.1
+G_ADI_AZIMUTH_INDICATOR+	Flight Info	DI.6
+G_ADI_ELEV_AVAIL+	Flight Info	DI.6
+G_ANGLE_OF_ATTACK+	AOA	DI.2
+G_ARP_INTERVAL+	WRS	DI.20
+G_ARP_PAIRS+	WRS	DI.20
+G_ARP_QUANTITY+	WRS	DI.20
+G_AUDIBLE_SIGNAL+	Audible Signal	DI.3
+G_AUTOCAL_INDICATOR+	Visual Indicators	DI.17
+G_AUTOCAL_TOGGLE+	Switch Bank	DI.15
+G_BORESIGHT_ANGLE+	WCM	DI.19
+G_B1+	WCM	DI.19
+G_COARSE_ROTATING+	IMS	DI.9
+G_COMPUTER_FAIL_SIGNAL+	Computer Fail Signal	DI.4
+G_CR2+	WCM	DI.19
+G_CR3+	WCM	DI.19
+G_CR4+	WCM	DI.19
+G_CR5+	WCM	DI.19
+G_CR6+	WCM	DI.19
+G_CR7+	WCM	DI.19
+G_CR8+	WCM	DI.19
+G_CR9+	WCM	DI.19
+G_CR10+	WCM	DI.19

Function Name	Device Interface	Section
+G_CR11+	WCM	DI.19
+G_CR12+	WCM	DI.19
+G_CR13+	WCM	DI.19
+G_DME_DISPLAY+	Flight Info	DI.6
+G_DME_FLAG+	Flight Info	DI.6
+G_DRS_DRIFT_ANGLE+	DRS	DI.5
+G_DRS_GROUND_SPEED+	DRS	DI.5
+G_DRS_MODE+	DRS	DI.5
+G_DRS_RELIABILITY+	DRS	DI.5
+G_DRS_TEST_RESULT+	DRS	DI.5
+G_E_COARSE_VEL_BIAS+	ISMR	DI.9R
+G_E_COARSE_VEL_SCALE+	IMSR	DI.9R
+G_E_FINE_VEL_BIAS	IMSR	DI.9R
+G_E_FINE_VEL_SCALE	IMSR	DI.9R
+G_E_LIGHT+	Panel	DI.10
+G_ENTER_LIGHT+	Panel	DI.10
+G_FLARE_FLAG+	WCM	DI.19
+G_FLR_AZIMUTH_CURSOR+	FLR	DI.7
+G_FLR_CURSOR_LIMITS+	FLR	DI.7
+G_FLR_DIRECTION+	FLR	DI.7
+G_FLR_DIRECTION_LIMIT+	FLR	DI.7
+G_FLR_LOCKED_ON+	FLR	DI.7
+G_FLR_MODE+	FLR	DI.7
+G_FLR_RANGE+	FLR	DI.7
+G_FLR_RANGE_CURSOR+	FLR	DI.7
+G_FLY_TO_NUM_SELECTOR+	Switch Bank	DI.15
+G_FLY_TO_SELECTOR+	Switch Bank	DI.15
+G_GUN_ENABLE+	WRS	DI.20
+G_HIGH_DRAG+	WRS	DI.20
+G_HMAX+	WCM	DI.19
+G_HMIN+	WCM	DI.19
+G_HSI_POINTER_1+	Flight Info	DI.6
+G_HSI_POINTER_2+	Flight Info	DI.6
+G_HUD_ALT_DISPLAY+	HUD	DI8
+G_HUD_AS_MODE+	HUD	DI.8
+G_HUD_AS_POSITION+	HUD	DI.8
+G_HUD_ASL_MODE+	HUD	DI.8
+G_HUD_ASL_POSITION+	HUD	DI.8
+G_HUD_FLTDIR_MODE+	HUD	DI.8
+G_HUD_FLTDIR_POSITION+	HUD	DI.8
+G_HUD_FPM_MODE+	HUD	DI.8
+G_HUD_FPM_POSITION+	HUD	DI.8

Function Name	Device Interface	Section
+G_HUD_HEADING_DISPLAY+	HUD	DI.8
+G_HUD_LSC_MODE+	HUD	DI.8
+G_HUD_LSC_POSITION+	HUD	DI.8
+G_HUD_NACC_DISPLAY+	HUD	DI.8
+G_HUD_PITCH_DISPLAY+	HUD	DI.8
+G_HUD_PUAC_MODE+	HUD	DI.8
+G_HUD_PUAC_POSITION+	HUD	DI.8
+G_HUD_PUC_MODE+	HUD	DI.8
+G_HUD_RELIABILITY+	HUD	DI.8
+G_HUD_RNGCUE_MODE+	HUD	DI.8
+G_HUD_ROLL_DISPLAY+	HUD	DI.8
+G_HUD_TEST_MODE+	HUD	DI.8
+G_HUD_USC_MODE+	HUD	DI.8
+G_HUD_USC_POSITION+	HUD	DI.8
+G_HUD_VERTVEL_DISPLAY+	HUD	DI.8
+G_HUD_VVFPM_MODE+	HUD	DI.8
+G_HUD_WARN_MODE+	HUD	DI.8
+G_IMS_E_VELOCITY+	IMS	DI.9
+G_IMS_ENABLE	IMS	DI.9
+G_IMS_MAG_HEADING+	IMS	DI.9
+G_IMS_MAG_VARIATION+	IMS	DI.9
+G_IMS_MODE+	IMS	DI.9
+G_IMS_N_VELOCITY+	IMS	DI.9
+G_IMS_PITCH+	IMS	DI.9
+G_IMS_ROLL+	IMS	DI.9
+G_IMS_SCALE+	IMS	DI.9
+G_IMS_STATUS+	IMS	DI.9
+G_IMS_TRUE_HEADING+	IMS	DI.9
+G_IMS_V_VELOCITY+	IMS	DI.9
+G_KEYBD_INPUT+	Panel	DI.10
+G_KEYBD_ENTER_LIGHT+	Panel	DI.10
+G_LAT_EJECTION_SPEED+	WCM	DI.19
+G_LAUNCH_SPEED+	WCM	DI.19
+G_LONG_EJECTION_SPEED+	WCM	DI.19
+G_LOWER_DECIMAL+	Panel	DI.10
+G_LOWER_FORMAT322+	Panel	DI.10
+G_MAGNUS_DEFLECTION+	WCM	DI.19
+G_MAP_DECENTER_TOGGLE+	Switch Bank	DI.15
+G_MAP_DISPLAYABLE+	PMDS	DI.11
+G_MAP_HOLD_TOGGLE+	Switch Bank	DI.15
+G_MAP_INDICATOR+	PMDS	DI.11
+G_MAP_LATITUDE_CT+	PMDSR	DI.11R
+G_MAP_LONGITUDE+	PMDSR	DI.11R
+G_MAP_NORTH_UP_TOGGLE+	Switch Bank	DI.15
+G_MAP_ORIENTATION+	PMDSR	DI.11R
+G_MAP_POINTER_ANGLE+	PMDS	DI.11
+G_MAP_POSITION+	PMDS	DI.11
+G_MAP_REFERENCE_PT+	PMDS	DI.11
+G_MAP_ROTATION+	PMDS	DI.11

ACCESS FUNCTION INDEX

Function Name	Device Interface	Section
+G_MAP_SCALE+	PMDS	DI.11
+G_MAP_SCALE_TOGGLE+	Switch Bank	DI.15
+G_MASTER_ARM+	WRS	DI.20
+G_MAX_RANGE+	WCM	DI.19
+G_MAX_USEFUL_RANGE+	WCM	DI.19
+G_MINE_FLAG+	WCM	DI.19
+G_MOTOR_BURN_TIME+	WCM	DI.19
+G_MRI_CLASS+	WCM	DI.19
+G_MULT_RACK+	WRS	DI.20
+G_MUZZLE_VEL+	WCM	DI.19
+G_N_COARSE_VEL_BIAS+	IMSR	DI.9R
+G_N_COARSE_VEL_SCALE+	IMSR	DI.9R
+G_N_FINE_VEL_BAIS	IMSR	DI.9R
+G_N_FINE_VEL_SCALE+	IMSR	DI.9R
+G_N_LIGHT+	Panel	DI.10
+G_PANEL_MODE_SELECTOR+	Switch Bank	DI.15
+G_PRES_POSITION_SELECTOR+	Switch Bank	DI.15
+G_RACK_DELAY+	WCM	DI.19
+G_RADAR_ALT+	Radar Altimeter	DI.12
+G_RC_CLASS+	WCM	DI.19
+G_RDY_STATIONS+	WRS	DI.20
+G_REL_IN_PROGRESS+	WRS	DI.20
+G_RELEASE_PULSE_WIDTH+	WCM	DI.19
+G_S_LIGHT+	Panel	DI.10
+G_SELF_TEST_TOGGLE+	Switch Bank	DI.15
+G_SINS_AGE+	SINS	DI.13
+G_SINS_EAST_VEL+	SINS	DI.13
+G_SINS_HEADING+	SINS	DI.13
+G_SINS_NORTH_VEL+	SINS	DI.13
+G_SINS_PITCH+	SINS	DI.13
+G_SINS_POSITION+	SINS	DI.13
+G_SINS_ROLL+	SINS	DI.13
+G_SINS_VALIDITY+	SINS	DI.13
+G_SLEW_RIGHT_LEFT+	Slew Control	DI.14
+G_SLEW_UP_DOWN+	Slew Control	DI.14
+G_TACAN_BEARING+	TACAN	DI.16
+G_TACAN_RANGE+	TACAN	DI.16
+G_U+	WCM	DI.19
+G_U_CORRECTED+	WCM	DI.19
+G_UPPER_FORMAT222+	Panel	DI.10
+G_UPPER_FORMAT321+	Panel	DI.10
+G_V_VEL_BIAS	IMSR	DI.9R
+G_V_VEL_SCALE+	IMSR	DI.9R
+G_W_LIGHT+	Panel	DI.10

Function Name	Device Interface	Section
+S_HUD_FLTDIR_MODE+	HUD	DI.8
+S_HUD_FLTDIR_POSITION+	HUD	DI.8
+S_HUD_FPM_MODE+	HUD	DI.8
+S_HUD_FPM_POSITION+	HUD	DI.8
+S_HUD_HEADING_DISPLAY+	HUD	DI.8
+S_HUD_LSC_MODE+	HUD	DI.8
+S_HUD_LSC_POSITION+	HUD	DI.8
+S_HUD_NACC_DISPLAY+	HUD	DI.8
+S_HUD_PITCH_DISPLAY+	HUD	DI.8
+S_HUD_PUAC_MODE+	HUD	DI.8
+S_HUD_PUAC_POSITION+	HUD	DI.8
+S_HUD_PUC_MODE+	HUD	DI.8
+S_HUD_ROLL_DISPLAY+	HUD	DI.8
+S_HUD_RNGCUE_MODE+	HUD	DI.8
+S_HUD_TEST_MODE+	HUD	DI.8
+S_HUD_USC_MODE+	HUD	DI.8
+S_HUD_USC_POSITION+	HUD	DI.8
+S_HUD_VERTVEL_DISPLAY+	HUD	DI.8
+S_HUD_VVFPM_MODE+	HUD	DI.8
+S_HUD_WARN_MODE+	HUD	DI.8
+S_IMS_ENABLE+	IMS	DI.9
+S_IMS_E_VELOCITY+	IMS	DI.9
+S_IMS_N_VELOCITY+	IMS	DI.9
+S_IMS_SCALE+	IMS	DI.9
+S_IMS_V_VELOCITY+	IMS	DI.9
+S_KEYBD_ENTER_LIGHT+	Panel	DI.10
+S_LOWER_DECIMAL+	Panel	DI.10
+S_LOWER_FORMAT322+	Panel	DI.10
+S_LOWER_WINDOW+	Panel	DI.10
+S_MAP_INDICATOR+	PMDS	DI.11
+S_MAP_LATITUDE_CT+	PMDSR	DI.11R
+S_MAP_LONGITUDE+	PMDSR	DI.11R
+S_MAP_ORIENTATION+	PMDSR	DI.11R
+S_MAP_POINTER_ANGLE+	PMDS	DI.11
+S_MAP_POSITION+	PMDS	DI.11
+S_MAP_REFERENCE_PT+	PMDS	DI.11
+S_MAP_ROTATION+	PMDS	DI.11
+S_MAP_SCALE+	PMDS	DI.11
+S_MARK_WINDOW+	Panel	DI.10
+S_N_COARSE_VEL_BIAS+	IMSR	DI.9R
+S_N_COARSE_VEL_SCALE+	IMSR	DI.9R
+S_N_FINE_VEL_BAIS	IMSR	DI.9R

Function Name	Device Interface	Section
+S_N_FINE_VEL_SCALE+	IMSR	DI.9R
+S_N_LIGHT+	Panel	DI.10
+S_NON_ALIGN_BLINK_RATE+	Visual Indicators	DI.17
+S_NON_ALIGN_INDICATOR+	Visual Indicators	DI.17
+S_S_LIGHT+	Panel	DI.10
+S_UPPER_FORMAT222+	Panel	DI.10
+S_UPPER_FORMAT321+	Panel	DI.10
+S_UPPER_WINDOW+	Panel	DI.10
+S_V_VEL_BIAS	IMSR	DI.9R
+S_V_VEL_SCALE+	IMSR	DI.9R
+S_W_LIGHT+	Panel	DI.10
+S_X_COARSE_ROTATION+	IMS	DI.9
+S_X_FINE_ROTATION+	IMS	DI.9
+S_X_GYRO_DRIFTRATE+	IMSR	DI.9R
+S_X_GYRO_SCALE+	IMSR	DI.9R
+S_Y_COARSE_ROTATION+	IMS	DI.9
+S_Y_FINE_ROTATION+	IMS	DI.9
+S_Y_GYRO_DRIFTRATE+	IMSR	DI.9R
+S_Y_GYRO_SCALE+	IMSR	DI.9R
+S_Z_COARSE_ROTATION+	IMS	DI.9
+S_Z_FINE_ROTATION+	IMS	DI.9
+S_Z_GYRO_DRIFTRATE+	IMSR	DI.9R
+S_Z_GYRO_SCALE+	IMSR	DI.9R
+SCALES_EQUAL+	PMDS	DI.11
+SCALES_GREATER+	PMDS	DI.11
+SCALES_LESSTHAN+	PMDS	DI.11
+SLEW_FLR_CURSORS+	FLR	DI.7
+SLEW_HUD_AS+	HUD	DI.8
+SLEW_MAP+	PMDS	DI.11
+START_DRS+	DRS	DI.5
+START_SINS+	SINS	DI.13
+STOP_DRS+	DRS	DI.5
+STOP_SINS+	SINS	DI.13
+TEST_ADC+	ADC	DI.1
+TEST_FLR+	FLR	DI.7
+WARMUP_WEAPON+	WRS	DI.20

DI.A2 APPENDIX 2

DICTIONARY TERM INDEX

Entry Name	Device Interface	Section
!+Active stations+!	WRS	DI.20
!+ADC alt valid+!	ADC	DI.1
!+ADC mach valid+!	ADC	DI.1
!+ADC TAS valid+!	ADC	DI.1
!+ADC test result+!	ADC	DI.1
!+ADI az+!	Flight Info	DI.6
!+ADI elev avail+!	Flight Info	DI.6
!+alt ADC+!	ADC	DI.1
!+alt RADAR+!	RA	DI.12
!+AOA+!	AOA	DI.2
!+AOA valid+!	AOA	DI.2
!+ARP Interval+!	WRS	DI.20
!+ARP Interval changed+!	WRS	DI.20
!+ARP Pairs+!	WRS	DI.20
!+ARP Pairs changed+!	WRS	DI.20
!+ARP Quantity+!	WRS	DI.20
!+ARP Quantity changed+!	WRS	DI.20
!+AS azimuth+!	HUD	DI.8
!+AS elevation+!	HUD	DI.8
!+ASL azimuth+!	HUD	DI.8
!+ASL elevation+!	HUD	DI.8
!+ASL rotation+!	HUD	DI.8
!+Aud signal+!	Audible Signal	DI.3
!+Auto-cal+!	Visual Indicators	DI.17
!+Auto-cal sw+!	Switch Bank	DI.15
!+Auto-cal sw changed+!	Switch Bank	DI.15
!+Az cursor+!	FLR	DI.7
!+Az cursor lft max+!	FLR	DI.7
!+Az cursor rgt max+!	FLR	DI.7
!+A1+!	WCM	DI.19
!+A2+!	WCM	DI.19
!+A3+!	WCM	DI.19
!+A4+!	WCM	DI.19
!+A5+!	WCM	DI.19
!+A6+!	WCM	DI.19
!+A7+!	WCM	DI.19
!+boresight angle+!	WCM	DI.19
!+B1+!	WCM	DI.19
!+central long+!	PMDSR	DI.11R
!+Comp Fail+!	Computer Fail Signal	DI.4
!+CR2+!	WCM	DI.19
!+CR3+!	WCM	DI.19
!+CR4+!	WCM	DI.19
!+CR5+!	WCM	DI.19
!+CR6+!	WCM	DI.19
!+CR7+!	WCM	DI.19
!+CR8+!	WCM	DI.19
!+CR9+!	WCM	DI.19

Entry Name	Device Interface	Section
!+HMax+!	WCM	DI.19
!+HMin+!	WCM	DI.19
!+HSI 1+!	Flight Info	DI.6
!+HSI 2+!	Flight Info	DI.6
!+horiz magnus deflection+!	WCM	DI.19
!+HUD alt+!	HUD	DI.8
!+HUD heading+!	HUD	DI.8
!+HUD NACC+!	HUD	DI.8
!+HUD pitch+!	HUD	DI.8
!+HUD reliable+!	HUD	DI.8
!+HUD roll+!	HUD	DI.8
!+HUD vertvel+!	HUD	DI.8
!+IMS mode+!	IMS	DI.9
!+IMS mode changed+!	IMS	DI.9
!+IMS ready+!	IMS	DI.9
!+IMS rel+!	IMS	DI.9
!+IMS rotating+!	IMS	DI.9
!+Keybd enter light+!	Panel	DI.10
!+Keybd input+!	Panel	DI.10
!+Keybd pressed+!	Panel	DI.10
!+L-probe+!	ADCR	DI.1R
!+lateral eject speed+!	WCM	DI.19
!+launch speed+!	WCM	DI.19
!+longitudinal eject speed+!	WCM	DI.19
!+low lat ct+!	PMDSR	DI.11R
!+LSC azimuth+!	HUD	DI.8
!+LSC elevation+!	HUD	DI.8
!+mach ADC+!	ADC	DI.1
!+magvar IMS+!	IMS	DI.9
!+Map decenter+!	Switch Bank	DI.15
!+Map decenter changed+!	Switch Bank	DI.15
!+Map displayable+!	PMDS	DI.11
!+Map hold+!	Switch Bank	DI.15
!+Map hold changed+!	Switch Bank	DI.15
!+Map indicator+!	PMDS	DI.11
!+Map latitude+!	PMDS	DI.11
!+Map longitude+!	PMDS	DI.11
!+Map north-up+!	Switch Bank	DI.15
!+Map north-up changed+!	Switch Bank	DI.15
!+Map orient+!	PMDSR	DI.11R
!+Map pointer angle+!	PMDS	DI.11
!+Map position valid+!	PMDS	DI.11
!+Map ref pt+!	PMDS	DI.11
!+Map rotation+!	PMDS	DI.11
!+Map scale+!	PMDS	DI.11
!+Map scale sw+!	Switch Bank	DI.15
!+Map scale sw changed+!	Switch Bank	DI.15

Entry Name	Device Interface	Section
!+Mark pressed+!	Panel	DI.10
!+Master Arm+!	WRS	DI.20
!+Master Arm changed+!	WRS	DI.20
!+max range+!	WCM	DI.19
!+max useful range+!	WCM	DI.19
!+Mine flag+!	WCM	DI.19
!+Motor burn time+!	WCM	DI.19
!+MRI class+!	WCM	DI.19
!+Mult rack+!	WRS	DI.20
!+muzzle velocity+!	WCM	DI.19
!+N coarse bias+!	IMSR	DI.9R
!+N coarse scale+!	IMSR	DI.9R
!+N fine bias+!	IMSR	DI.9R
!+N fine scale+!	IMSR	DI.9R
!+N light+!	Panel	DI.10
!+N vel IMS+!	IMS	DI.9
!+Non-align+!	Visual Indicators	DI.17
!+Panel mode+!	Switch Bank	DI.15
!+Panel mode changed+!	Switch Bank	DI.15
!+pitch IMS+!	IMS	DI.9
!+Pres pos+!	Switch Bank	DI.15
!+Pres pos changed+!	Switch Bank	DI.15
!+PUAC azimuth+!	HUD	DI.8
!+PUAC elevation+!	HUD	DI.8
!+Ready Stations+!	WRS	DI.20
!+Rel in Progress+!	WRS	DI.20
!+RA valid+!	RA	DI.12
!+RE pressed+!	WRS	DI.20
!+Rng cursor+!	FLR	DI.7
!+Rng cursor max+!	FLR	DI.7
!+Rng cursor min+!	FLR	DI.7
!+roll IMS+!	IMS	DI.9
!+release pulse width+!	WCM	DI.19
!+S light+!	Panel	DI.10
!+scales equal+!	PMDS	DI.11
!+scales greater+!	PMDS	DI.11
!+scales lessthan+!	PMDS	DI.11
!+Self-test+!	Switch Bank	DI.15
!+Self-test changed+!	Switch Bank	DI.15
!+SINS attitude age+!	SINS	DI.13
!+SINS attitude valid+!	SINS	DI.13
!+SINS east vel+!	SINS	DI.13
!+SINS heading+!	SINS	DI.13
!+SINS lat+!	SINS	DI.13
!+SINS long+!	SINS	DI.13
!+SINS north vel+!	SINS	DI.13
!+SINS pitch+!	SINS	DI.13
!+SINS position age+!	SINS	DI.13
!+SINS position valid+!	SINS	DI.13

DI.A3 APPENDIX 3

EVENT INDEX

Event	Device Interface	Section
@F(!+ADC alt valid+!)	ADC	DI.1
@F(!+ADC mach valid+!)	ADC	DI.1
@F(!+ADC TAS valid+!)	ADC	DI.1
@F(!+ADI elev avail+!)	Flight Info	DI.6
@F(!+AOA valid+!)	AOA	DI.2
@F(!+DRS reliable+!)	DRS	DI.5
@F(!+Enter pressed+!)	Panel	DI.10
@F(!+FLR in terrain following+!)	FLR	DI.7
@F(!+FLR locked on +!)	FLR	DI.7
@F(!+HUD reliable+!)	HUD	DI.8
@F(!+IMS rel+!)	IMS	DI.9
@F(!+IMS ready+!)	IMS	DI.9
@F(!+Keybd pressed+!)	Panel	DI.10
@F(!+Mark pressed+!)	Panel	DI.10
@F(!+RA valid +!)	Radar Alt	DI.12
@F(!+RE pressed+!)	WRS	DI.20
@F(!+SINS attitude valid+!)	SINS	DI.13
@F(!+SINS position valid+!)	SINS	DI.13
@F(!+SINS velocity valid+!)	SINS	DI.13
@F(!+Slew Displacement non-zero+!)	Slew Control	DI.14
@F(!+TAC bearing valid+!)	TACAN	DI.16
@F(!+TAC range valid+!)	TACAN	DI.16
@F(!+TD pressed+!)	WRS	DI.20
@T(!+ADC alt valid+!)	ADC	DI.1
@T(!+ADC mach valid+!)	ADC	DI.1
@T(!+ADC TAS valid+!)	ADC	DI.1
@T(!+ADI elev avail+!)	Flight Info	DI.6
@T(!+ARP Interval changed+!)	WRS	DI.20
@T(!+ARP Pairs changed+!)	WRS	DI.20
@T(!+ARP Quantity changed+!)	WRS	DI.20
@T(!+AOA valid+!)	AOA	DI.2
@T(!+Auto-cal changed+!)	Switch Bank	DI.15
@T(!+DRS reliable+!)	DRS	DI.5
@T(!+DRS mode changed+!)	DRS	DI.5
@T(!+Enter pressed!+)	Panel	DI.10
@T(!+FLR in terrain following+!)	FLR	DI.7
@T(!+FLR locked on+!)	FLR	DI.7
@T(!+Fly to num changed+!)	Switch Bank	DI.15
@T(!+Fly to state changed+!)	Switch Bank	DI.15
@T(!+Gun Enable changed+!)	WRS	DI.20
@T(!+High Drag changed+!)	WRS	DI.20
@T(!+HUD reliable+!)	HUD	DI.8
@T(!+IMS mode changed+!)	IMS	DI.9
@T(!+IMS rel+!)	IMS	DI.9
@T(!+IMS ready+!)	IMS	DI.9

Event	Device Interface	Section
@T(!+Keybd input ready+!)	Panel	DI.10
@T(!+Keybd pressed+!)	Panel	DI.10
@T(!+Map hold changed+!)	Switch Bank	DI.15
@T(!+Map north up changed+!)	Switch Bank	DI.15
@T(!+Map scale sw changed+!)	Switch Bank	DI.15
@T(!+Mark pressed!+)	Panel	DI.10
@T(!+Panel mode changed+!)	Switch Bank	DI.15
@T(!+Pres pos changed+!)	Switch Bank	DI.15
@T(!+RA valid+!)	Radar Alt	DI.12
@T(!+RE pressed+!)	WRS	DI.20
@T(!+Self-test changed+!)	Switch Bank	DI.15
@T(!+SINS attitude valid+!)	SINS	DI.13
@T(!+SINS position valid+!)	SINS	DI.13
@T(!+SINS velocity valid+!)	SINS	DI.13
@T(!+Slew Displacement non-zero+!)	Slew Control	DI.14
@T(!+TAC bearing valid+!)	TACAN	DI.16
@T(!+TAC range valid+!)	TACAN	DI.16
@T(!+TD pressed+!)	WRS	DI.20
@T(!+Update changed+!)	Switch Bank	DI.15
@T(!+WAYPT available+!)	WIS	DI.18
@T(!+Weapon mode changed+!)	WRS	DI.20

INDEX TO SYSTEM GENERATION PARAMETERS
(With expected values)

Parameter Name	Device Interface	Expected Value
¢ADC_ALT_MAX¢	ADC	50,000 feet
¢ADC_ALT_MIN¢	ADC	-1,000 feet
¢ADC_ALT_RES¢	ADC	14 feet
¢ADC_MACH_RES¢	ADC	.0006
¢ADC_TAS_RES¢	ADC	.4 feet/sec
¢ADI_AZIMUTH_MAX¢	Flight Info	3 degrees
¢ADI_ELEVATION_MAX¢	Flight Info	20 degrees
¢ADI_RES¢	Flight Info	.1 degrees
¢AOA_RES¢	AOA	.18 degrees
¢ARP_INTERVAL_MAX¢	WRS	1000
¢ARP_QUANTITY_MAX¢	WRS	100
¢BEEP_FREQ¢	Audible Signal	1/sec
¢DRIFT_ANGLE_RES¢	DRS	.005 degrees
¢FLR_MIN_RANGE¢	FLR	1000 feet
¢FLR_MAX_RANGE¢	FLR	50,000 feet
¢FLR_RANGE_RES¢	FLR	20 feet
¢FLR_MAX_ELEV¢	FLR	20 degrees
¢FLR_ELEV_RES¢	FLR	.1 degrees
¢FLR_MAX_AZIMUTH¢	FLR	40 degrees
¢FLR_AZIMUTH_RES¢	FLR	.1 degrees
¢FLR_MAX_RANGE_CURSOR¢	FLR	120,000 feet
¢FLR_MIN_RANGE_CURSOR¢	FLR	0 feet
¢FLR_RNGCUR_RES¢	FLR	40 feet
¢FLR_MAX_AZ_CURSOR¢	FLR	45 degrees
¢FLR_AZCUR_RES¢	FLR	.1 degree
¢FLY_TO_MAX¢	Switch Bank	10
¢GND_SPEED_RES¢	DRS	.1 knot
¢HSI_RES¢	Flight Info	.1 degrees
¢HUD_ALT_MIN¢	HUD	-1,000 feet
¢HUD_ALT_MAX¢	HUD	60,000 feet
¢HUD_ALT_RES¢	HUD	10 feet
¢HUD_ANGULAR_RES¢	HUD	.2 degrees
¢HUD_BLINK_RATE¢	*HUD*	*1/ sec*
¢HUD_FLTDIR_MAX_AZ¢	HUD	8 degrees
¢HUD_SLEW_RATE¢	HUD	4 arc-sec/sec
¢HUD_SYMBOL_MAX_AZ¢	HUD	16 degrees
¢HUD_SYMBOL_MAX_EL¢	HUD	16 degrees
¢HUD_VERTACC_MAX¢	HUD	10,000 feet/min^2
¢HUD_VERTVEL_MAX¢	HUD	10,000 feet/min
¢HUD_VERTACC_RES¢	HUD	4 feet/min^2
¢HUD_VERTVEL_RES¢	HUD	4 feet/min

Parameter Name	Device Interface	Expected Value
¢IMS_ATTITUDE_RES¢	IMS	.1 degree
¢IMS_COARSE_ROT_RES¢	IMS	.01 degrees
¢IMS_COARSE_VEL_RES¢	IMS	.01 feet/sec
¢IMS_FINE_ROT_RES¢	IMS	.0001 degree
¢IMS_FINE_VEL_RES¢	IMS	.0001 feet/sec
¢IMSR_COARSE_BIAS_MAX¢	IMS (reconfiguration)	.1
¢IMSR_COARSE_BIAS_MIN¢	IMS (reconfiguration	- .1
¢IMSR_COARSE_BIAS_RES¢	IMS (reconfiguration)	.0003
¢IMSR_COARSE_VSCALE_MAX	IMS (reconfiguration)	.038
¢IMSR_COARSE_VSCALE_MIN	IMS (reconfiguration)	.026
¢IMSR_COARSE_VSCALE_RES	IMS (reconfiguration)	.00003
¢IMSR_CORR_INC_MAX¢	IMS (reconfiguration)	.48
¢IMSR_CORR_INC_MIN¢	IMS (reconfiguration)	.32
¢IMSR_CORR_INC_RES¢	IMS (reconfiguration)	.0004
¢IMSR_DRIFT_MAX¢	IMS (reconfiguration)	1
¢IMSR_DRIFT_MIN¢	IMS (reconfiguration)	-1
¢IMSR_DRIFT_RES¢	IMS (reconfiguration)	.001
¢IMSR_FINE_BIAS_MAX¢	IMS (reconfiguration)	.01
¢IMSR_FINE_BIAS_MIN¢	IMS (reconfiguration)	- .01
¢IMSR_FINE_BIAS_RES¢	IMS (reconfiguration)	.0003
¢IMSR_FINE_VSCALE_MAX¢	IMS (reconfiguration)	.00038
¢IMSR_FINE_VSCALE_MIN¢	IMS (reconfiguration)	.00026
¢IMSR_FINE_VSCALE_RES¢	IMS (reconfiguration)	.000001
¢MAP_LAT_RES¢	PMDS	
¢MAP_LEGEND_RES¢	PMDS	1 degree
¢MAP_LONG_RES¢	PMDS	
¢MAP_POINTER_RES¢	PMDS	1 degree
¢MAP_ROT_RES¢	PMDS	1 degree
¢MAP_SCALE_ARRAY¢	PMDS	
¢NUM_MAP_SCALES¢	PMDS	2
¢NUM_WEAP_STATIONS¢	WRS	6
¢NUM_WEAP_TYPES¢	WRS	100
¢RADAR_ALT_MAX¢	RADAR ALT	5000 feet
¢RADAR_ALT_MIN¢	RADAR ALT	10 feet
¢RADAR_ALT_RES¢	RADAR ALT	1 foot
¢SINS_ATTITUDE_RES¢	SINS	.005 degree
¢SINS_LAT_RES¢	SINS	.001 degree
¢SINS_LONG_RES¢	SINS	.000001 degrees
¢SLEW_MIN¢	SLEW Control	.148
¢SLEW_MAX¢	SLEW Control	4
¢SLEW_RES¢	SLEW Control	.001
¢TACAN_BEARING_RES¢	TACAN	.25 degrees
¢TACAN_RANGE_MAX¢	TACAN	400 nmi
¢TACAN_RANGE_RES¢	TACAN	.025 nmi

MAPPING FROM REQUIREMENTS TO ACCESS FUNCTIONS

The table in this section shows how to control various output data items using the access functions provided by this module. This table is intended to help a programmer choose the correct functions to implement the requirements, which are expressed in terms of data items.

The table is organized in the same order as section 4 of the requirements document (reference (1)). Each entry in the table includes

1) the number of the subsection in section 4 of the requirements where the associated software function is described; it is expected that the programmer will be working from this description, looking for the correct access function to call to fulfill the requirement;

2) the output data item names used in the software function description;

3) the name(s) of associated access functions;

4) the section number in this report where those access functions are described.

Where it is not obvious, the table also shows how output data item values map to access function parameters. This is done in one of the following two ways:

1) //AUTOCAL//:=On/Off +S_AUTOCAL_INDICATOR+(On/Off)

   Interpretation:
   To set //AUTOCAL// to On, call +S_AUTOCAL_INDICATOR with On
   To set //AUTOCAL// to Off, call +S_AUTOCAL_INDICATOR with Off

2) //CURAZCOS//:= cosine(a) +S_FLR_AZIMUTH_CURSOR+(a)
   //CURAZSIN//:= sine(a)

   Interpretation:
   The access function takes two parameters, an angle "a" and another parameter that is not relevant for these data items. To assign to //CURAZCOS// the cosine of angle "a" and to //CURAZSIN// the sine, call +S_FLR_AZIMUTH_CURSOR+ with the angle "a"; the device interface module will calculate the sine and cosine for the data items.

The parameter values are not given in some cases where the reader can determine the correct values by reading the device interface module specification. Thus for the following example, the parameters for the function are omitted

Set azimuth cursor out of view	+REMOVE_FLR_AZIMUTH_CURSOR+

Legend of symbols used in the table:

a, b, c	quantities of type angle
d	quantity of type distance
g	quantity of type acceleration
s	quantity of type speed
true, false	values of type logical
\$. . .\$	value defined for data item or parameter by enumeration

Req. Section	Data item(s)	Function(s)	DI Section
4.1.1	//AUTOCAL//:=On/Off	+S_AUTOCAL_INDICATOR+(On/Off)	DI-17
4.1.2	//COMPCTR//:=On/Off	+S_IMS_ENABLE+(true/false)	DI-9
4.1.3	//COMPFAIL//:=On	+S_COMPUTER_FAIL_SIGNAL+(true)	DI-4
4.1.4	To torque an axis angle "a":		
	//XGYCOM//	+S_X_FINE_ROTATION+(a)	DI-9
	//YGYCOM//	+S_Y_FINE_ROTATION+(a)	DI-9
	//ZGYCOM//	+S_Z_FINE_ROTATION+(a)	DI-9
4.1.5	//IMSSCAL//:=$Fine$/$Coarse$	+S_IMS_SCALE+($Fine$/$Coarse$)	DI-9
4.1.6	To slew an axis angle "a"		
	//XSLEW//, //XSLSEN//	+S_X_COARSE_ROTATION+(a)	DI-9
	//YSLEW//, //YSLSEN//	+S_Y_COARSE_ROTATION+(a)	DI-9
	//ZSLEW//, //ZSLSEN//	+S_Z_COARSE_ROTATION+(a)	DI-9
4.1.7	//LATGT70//:=Yes/No	done by IMS module internally	
4.1.8.1	//IMSNA//:=On/Off	+S_NON_ALIGN_INDICATOR+(On/Off)	DI-17
4.1.8.2	Flash //IMSNA//	+S_NON_ALIGN_BLINK_RATE+(r) +S_NON_ALIGN_INDICATOR+ ($Intermittent$)	DI-17
4.2.1	//CURENABL//:=On/Off	+S_FLR_MODE+($CDCE$/$Idle$)	DI-7
4.2.2.1	//CURAZCOS//:=cosine(a) //CURAZSIN//:=sine(a)	+S_FLR_AZIMUTH_CURSOR+(a)	DI-7
	To set out of view	+REMOVE_FLR_AZIMUTH_CURSOR+	DI-7
4.2.2.2	//CURPOS//:= d	+S_FLR_RANGE_CURSOR+(d)	DI-7
4.2.3	//ANTSLAVE//:=On/Off	+S_FLR_MODE+($Ranging$/$Idle$)	DI-7
4.2.4	//STEEREL//:= a	+S_FLR_DIRECTION+(a, b) (for meaning of b see below)	DI-7
4.2.5	//STEERAZ//:= b	+S_FLR_DIRECTION+(a, b) done by FLR module internally when in TF mode	DI-7
4.2.6	//GNDTRVEL//:= s	done by FLR module (in TF mode) internally	

Req. Section	Data item(s)	Function(s)	DI Section
4.2.7	//FPANGL//:= a (to FLR)	done by FLR module (in TF mode)	
	//FPANGL//:= a (to ADI needle)	+S_ADI_ELEV_INDICATOR(a)	DI-6
	ADI needle out of view	+REMOVE_ADI_ELEV_INDICATOR+	DI-6
4.3.1	//HUDAS//:=On/Off	+S_HUD_AS_MODE+(On/Off)	DI-8
	//ASEL//:= a //ASAZ//:= b	+S_HUD_AS_POSITION+(a, b)	DI-8
	To position AS in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
	To slew AS, see also	+SLEW_HUD_AS+	DI-8
4.3.2	//HUDASL//:=On/Off	+S_HUD_ASL_MODE+(On/Off)	DI-8

	//ASLEL//:= a
	//ASLEL//:= b
	//ASLCOS//:= cosine(c)
	//ASLSIN//:= sine(c)
		+S_HUD_ASL_POSITION+(a, b, c)	DI-8

	To position ASL in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
4.3.3	//BAROHUD//:= d	+S_HUD_ALT_DISPLAY+(d)	DI-8
4.3.4	//FLTDIRAZ//:= a	+S_HUD_FLTDIR_POSITION+(a)	DI-8
	To position FLTDIR in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
4.3.5.1	//FPMEL//:= a //FPMAZ//:= b	+S_HUD_FPM_POSITION+(a, b)	DI-8
	To position FPM in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
4.3.5.2	//HUDFPM//:=$Blink$/$Constant$	+S_HUD_FPM_MODE+($Blink$/On)	DI-8
4.3.6	//MAGHDGH//:= a	+S_HUD_HEADING_DISPLAY+(a)	DI-8
4.3.7	//PTCHANG//:= a	+S_HUD_PITCH_DISPLAY+(a)	DI-8

Req. Section	Data item(s)	Function(s)	DI Section
4.3.8.1	PUAC in view/out of view	+S_HUD_PUAC_MODE+(On/Off)	DI-8
	//PUACEL//:= a	+S_HUD_PUAC_POSITION+(a, b)	DI-8
	//PUACAZ//:= b		
	To position PUAC in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
4.3.8.2	Flash PUAC	+S_HUD_PUAC_MODE+($Blink$)	DI-8
4.3.9	//HUDPUC//:=On/Off	+S_HUD_PUC_MODE+($Blink$/Off)	DI-8
4.3.10	//ROLLCOSH//:=cosine(a)	+S_HUD_ROLL_DISPLAY+(a)	DI-8
	//ROLLSINH//:=sine(a)		
4.3.11.0	//HUDSCUE//:=On/Off	+S_HUD_LSC_MODE+(On/Off)	DI-8
		+S_HUD_USC_MODE+(On/Off)	DI-8
4.3.11.1	//LSOLCUEL//:= a	+S_HUD_LSC_POSITION+(a, b)	DI-8
	//LSOLCUAZ//:= b		
	To position LSC in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
	To position LSC as range cue, see also	+S_HUD_RNGCUE_MODE+	DI-8
4.3.11.2	USC in view/out of view	+S_HUD_USC_MODE+(On/Off)	DI-8
	//USOLCUEL//:= a	+S_HUD_USC_POSITION+(a, b)	DI-8
	//USOLCUAZ//:= b		
	To position USC in test pattern, see also	+S_HUD_TEST_MODE+	DI-8
4.3.11.3	Flash solution cues	+S_HUD_LSC_MODE+($Blink$)	DI-8
		+S_HUD_USC_MODE+($Blink$)	DI-8
4.3.12.0	//HUDVEL//:=On/Off	+S_HUD_VVFPM_MODE+(On/Off)	DI-8
4.3.12.1	//VERTVEL//:= s	+S_HUD_VERTVEL_DISPLAY+(s)	DI-8
4.3.12.2	//VTVELAC//:= s (velocity)	+S_HUD_VERTVEL_DISPLAY+(s)	DI-8
	//VTVELAC//:= g (acceleration)	+S_HUD_ACCEL_DISPLAY+(g)	DI-8
4.4.1	//BMBREL//	+RELEASE_WEAPON+	DI-19
4.4.2	//FIRRDY//	either +RELEASE_WEAPON+ or +WARMUP_WEAPON+	DI-19
4.4.3	//BMBTON//:= On/Off	+S_AUDIBLE_SIGNAL+(On/Off)	DI-3

Req. Section	Data item(s)	Function(s)	DI Section
4.5.1	//AZRING//:= a	+S_MAP_LEGEND+(a)	DI-11
4.5.2	//DESTPNT//:= a	+S_MAP_POINTER_ANGLE+(a)	DI-11
4.5.3	//MAPOR//:= a	+S_MAP_ROTATION+(a)	DI-11
4.5.4	To position map, setting location, scale, and reference point, see //XCOMMC//, //XCOMMF// //YCOMM//	+S_MAP_POSITION+ +S_MAP_SCALE+ +S_MAP_REFERENCE_PT+	DI-11 DI-11 DI-11
	To position on hash marks	+DISPLAY_MAP_WARNING+	DI-11
4.6	//LWDIG1//, //LWDIG2//, //LWDIG3//, //LWDIG4//, //LWDIG5//, //LWDIG6//, //LWDIG7//	+CLEAR_LOWER_WINDOW+ +S_LOWER_WINDOW+	DI-10
	//UWDIG1//, //UWDIG2//, //UWDIG3//, //UWDIG4//, //UWDIG5//, //UWDIG6//	+CLEAR_UPPER_WINDOW+ +S_UPPER_WINDOW+	DI-10
	//ULIT222//	+S_FORMAT_U222+	DI-10
	//ULIT321//	+S_FORMAT_U321+	DI-10
	//LLITDEC//	+S_DECIMAL_PT+	DI-10
	//ULITN//, //ULITS//, //LLITE//, //LLITW//	+S_N_LIGHT+, +S_S_LIGHT+ +S_E_LIGHT+, +S_W_LIGHT+	DI-10
4.6.3	//MARKWIN//	+CLEAR_MARK+ +S_MARK_WINDOW+	DI-10
4.6.4	//ENTLIT//	+S_ENTER_LIGHT+	
4.7.1	//BRGDEST//:= a	+S_HSI_POINTER_1+(a)	DI-6
4.7.2	//GNDTRK//:= a	+S_HSI_POINTER_2+(a)	DI-6
4.7.3	To display distance d //RNGUNIT//, //RNGTEN//, //RNGHND//, //LFTDIG//	+S_DME_DISPLAY+(d) +S_DME_FLAG+(true/false)	DI-6
4.8.1	//STERROR//:= a	+S_ADI_AZIMUTH_INDICATOR+(a)	DI-6

DI.A6 APPENDIX 6

COORDINATE SYSTEMS

This section describes the three coordinate systems used in the specifications of the device interface modules. All four coordinate systems are right-handed systems. The three coordinate systems are:

(1) the airframe system, defined in terms of the airframe,

(2) the IMS system, defined in terms of the platform of the Inertial Measurement Set,

(3) the SINS system, defined in terms of the ship body.

Airframe system

The airframe coordinate system has axes Xa, Ya, and Za. The Ya axis lies long the A/C boresight line with the positive direction being forward (toward nose, from tail). The Xa axis points out in the direction of the right wing and is defined so that the Xa - Ya plane is horizontal and the Za axis points upward when the A/C wings are level and the A/C is right side up.

IMS system

The IMS coordinate system has axes Xp, Yp, and Zp. These are often referred to as North, East, and Down (vertical), respectively, because in normal operation the software attempts to keep them aligned in these directions. When the A/C is flying north and level, Xa aligns with Yp, Ya aligns with Xp, and Za aligns with Zp. The directions of Zp and Za are opposite.

SINS system

The SINS coordinate system has axes Xs, Ys, and Zs. The Ys axis is toward the forward end of the ship, the Xs axis is to the starboard side of the ship, and the Zs axis is vertical and points upward.

DI.A7 APPENDIX 7

NOTATION GUIDE

Notation	Meaning	Further Explanation
+...+	Access function name	DI.ii, section 2.1
p1,p2,..,pn	Parameter number 1, 2, ..., n	DI.ii, section 2.1
@T(...)	Event occurs when condition becomes true	DI.ii, section 2.2
@F(...)	Event occurs when condition becomes false	DI.ii, section 2.2
!*...*!	Virtual device modes	DI.ii, section 2.3
$...$	Symbolic value	DI.ii, section 5
!+..+!	Local dictionary entry name	DI.ii, section 6
%...%	Undesired event name	DI.ii, section 7
¢...¢	System generation time parameter	DI.ii, section 9